Package 'SelectionTools'

November 14, 2026

Version 26.6

Date 2026-11-13

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. Functions to fit linear mixed models and for phasing and imputation of DNA marker data are also included.

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 CRAN

Date/Publication 2026-11-14 08:20:32 UTC

Contents

append.population 10
be.quiet 11
concat.population 11
copy.population 12
cross 12
data.params 15

define.effects . . . 15
define.effects.df . . . 16
define.genome . . . 17
define.map . . . 18
deprecated . . . 20
dh . . . 20
divide.population . . . 21
effect.weight.set.all . . . 22
effmap.remove . . . 22
effmap.remove.all . . . 23
evaluate.all.loci . . . 23
evaluate.allele . . . 24
evaluate.allele.freq . . . 25
evaluate.allele.freq2 . . . 26
evaluate.allele2 . . . 27
evaluate.genome . . . 28
evaluate.genotype . . . 29
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 71
gs.esteff.lsq 72
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 81
gs.lambda.const 81
gs.lambda.emstep 82
gs.lambda.hsq 82

gs.lambda.reg . . . 83
gs.lambda.rmla . . . 84
gs.lambda.rmla.02 . . . 84
gs.lambda.rmlc . . . 85
gs.lambda.rmlr . . . 86
gs.lambda.rmlv . . . 86
gs.lambda.rrblup . . . 87
gs.make.A . . . 88
gs.mme.coeff . . . 88
gs.mme.invcoeff . . . 89
gs.mme.restcoeff . . . 89
gs.mme.restrhs . . . 90
gs.mme.rhs . . . 90
gs.mme.solve . . . 91
gs.mmet.coeff . . . 92
gs.mmet.coeff.3 . . . 92
gs.mmet.rhs . . . 93
gs.mmet.solve . . . 93
gs.neg.effall . . . 94
gs.plot.effects . . . 94
gs.plot.model.fit . . . 95
gs.plot.validation . . . 96
gs.pos.effall . . . 97
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 . . . 101
gs.set.lambda . . . 102
gs.set.num.threads . . . 102
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
influence.st.lmm . . . 114
info . . . 117
info.cat . . . 118
init.population . . . 118
lib00.map1 . . . 119
lib00.map1a . . . 120
lib00.map2 . . . 120
linkage.drag . . . 121
linkage.map.get . . . 121
linkage.map.load . . . 122
linkage.map.save . . . 123
list.effects . . . 124
list.populations . . . 124
load.effmap . . . 125
load.internal.effmap . . . 126
load.linkage.map . . . 126
mab.compare . . . 127
mab.df . . . 134
mab.Examples . . . 134
mab.Input.files . . . 135
mab.load.data . . . 136
mab.save.inds . . . 137
mab.simulate . . . 138
mab.tabulate . . . 142
masbc.input . . . 144
optimize.population . . . 144
ph.linkage.map.create . . . 145
phenotype.population . . . 146
plabsim . . . 147
plabsim.init . . . 147
plabsim.R.date . . . 147
plabsim.R.version . . . 148
plabsim.version . . . 148
plant . . . 148
population.append . . . 149
population.concat . . . 150
population.copy . . . 151
population.divide . . . 151
population.exist . . . 152
population.individual.remove . . . 153
population.info.get . . . 153
population.info.set . . . 154
population.list . . . 155
population.matrix.load . . . 155
population.matrix.save . . . 156

population.name.swap . . . 156
population.optimize . . . 157
population.plabsim.load . . . 157
population.plabsim.save . . . 158
population.remove . . . 158
population.remove.all . . . 159
population.rename . . . 160
population.resize . . . 160
population.sample . . . 161
population.size.get . . . 162
population.sort . . . 162
population.swap.name . . . 163
population.transfer . . . 164
propsd . . . 164
read.vcf . . . 166
remove.all.populations . . . 168
remove.effmaps . . . 169
remove.evaluate.population . . . 169
remove.genotype.population . . . 170
remove.map . . . 170
remove.population . . . 171
rename.population . . . 171
reset.all . . . 172
reset.mdp . . . 173
resize.population . . . 173
resources . . . 174
return.population . . . 174
rng.choose . . . 175
rng.info . . . 175
rng.init . . . 176
rng.list . . . 177
sample.population . . . 177
save.linkage.map . . . 178
sdev . . . 179
sel.target.define . . . 179
select.all.best . . . 180
select.all.best.intern . . . 181
select.genotypes . . . 182
select.n.best . . . 183
select.n.best.segments . . . 184
set.co.freq . . . 185
set.eff.weight . . . 185
set.genome.par . . . 186
set.info.level . . . 187
set.mdp . . . 187
set.NoLociInit . . . 188
set.population.gvalue . . . 188
set.population.info . . . 189

show.phase . . . 190
single.cross . . . 190
sm.start.timer . . . 191
sm.stop.timer . . . 191
splitdt . . . 192
ssd.mating . . . 192
st.calc.ld . . . 193
st.calc.ld.2 . . . 196
st.calc.q . . . 196
st.calc.rf . . . 197
st.calculate.enrichment . . . 198
st.chrom.stats . . . 200
st.compact.data.set . . . 200
st.copy.marker.data . . . 201
st.cross.allocate . . . 202
st.cross.constraints . . . 206
st.cross.evaluate . . . 208
st.cross.plan . . . 214
st.cross.plan.execute . . . 224
st.data.combine . . . 226
st.data.set.list . . . 227
st.data.set.remove . . . 229
st.data.set.remove.all . . . 230
st.datadir . . . 230
st.dataframe.to.STvcf . . . 231
st.dd . . . 233
st.def.hblocks . . . 234
st.destroy.phase . . . 235
st.detect.core . . . 236
st.diallel.cross . . . 238
st.genetic.distances . . . 239
st.genetic.distances.02 . . . 242
st.genetic.distances.fct . . . 242
st.get.A . . . 243
st.get.info.level . . . 244
st.get.map . . . 245
st.get.num.threads . . . 245
st.get.simpop . . . 247
st.get.simpop.perfdata . . . 248
st.get.simpop2 . . . 248
st.gwas . . . 249
st.hybrid.data.set . . . 256
st.id . . . 258
st.indir . . . 258
st.info . . . 259
st.keep.intersection.markers . . . 259
st.kernel.structure . . . 260
st.LDheatmap . . . 262

st.LDplot.ld . . . 266
st.LDplot.map . . . 266
st.lmm . . . 267
st.lmm.allfit . . . 309
st.lmm.bootstrap . . . 311
st.lmm.by . . . 315
st.lmm.compare . . . 317
st.lmm.components . . . 319
st.lmm.diagnostics . . . 324
st.lmm.model . . . 326
st.lmm.profile . . . 327
st.lmm.refit . . . 331
st.lmm.simulate . . . 333
st.load.performance.data . . . 337
st.load.vcf.data . . . 339
st.lsm . . . 342
st.lsm-hypotheses . . . 353
st.mark.alleles . . . 354
st.marker.data.statistics . . . 355
st.markerdata.to.STvcf . . . 356
st.ocs . . . 358
st.od . . . 360
st.oma . . . 361
st.outdir . . . 364
st.phase . . . 364
st.plot.corr . . . 368
st.plot.corr.l . . . 368
st.plot.gene.diversity . . . 369
st.plot.ggt . . . 370
st.plot.ggt.src . . . 372
st.predict . . . 373
st.read.map . . . 377
st.read.map.df . . . 379
st.read.marker.data . . . 380
st.read.marker.data.df . . . 381
st.read.performance.data . . . 383
st.recode.hbc . . . 384
st.recode.hil . . . 385
st.recode.ref . . . 386
st.recode.ref.2 . . . 387
st.reset . . . 387
st.restrict.marker.data . . . 388
st.return.performance.data . . . 389
st.select.phen . . . 390
st.set.hblocks . . . 391
st.set.info.level . . . 391
st.set.matrix.ops . . . 392
st.set.num.threads . . . 392

st.set.openblas.threads . . . 393
st.set.sim.ef . . . 394
st.set.sim.gp . . . 396
st.set.sim.mp . . . 397
st.set.sim.pp . . . 397
st.set.simpop . . . 398
st.start.timer . . . 400
st.stop.timer . . . 400
st.STvcf.to.dataframe . . . 401
st.switch.error . . . 402
st.write.map . . . 404
st.write.marker.data . . . 404
summarize.gvalue . . . 406
swap.population.name . . . 406
talk.to.me . . . 407
v-dentmaize-vcf . . . 407
v-dentmaize2-vcf . . . 408
v-flintmaize-vcf . . . 409
v-flintmaize2-vcf . . . 410
v-hybmaize-phe . . . 412
v-hybmaize2-phe . . . 413
v-lmm-climate . . . 414
v-lmm-corn . . . 414
v-lmm-data . . . 415
v-lmm-fertilizer-regression . . . 421
v-lmm-fertilizer-replicated . . . 421
v-lmm-halfdiallel . . . 422
v-lmm-iris . . . 423
v-lmm-iris-two . . . 424
v-lmm-machines . . . 425
v-lmm-mtcars . . . 426
v-lmm-rats . . . 427
v-lmm-tomato . . . 428
v-lmm-tropmaize . . . 428
v-lmm-tropmaize-lattice . . . 430
v-rapeseed . . . 431
v-tropmaize-map . . . 432
v-tropmaize-phe . . . 433
v-tropmaize-pop . . . 433
v-tropmaize-vcf . . . 434
write.vcf . . . 435
write.version.2 . . . 437

Index **438**

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

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
reset.all()
st.set.sim.gp(data.set = "maize.example")
st.set.sim.mp(data.set = "maize.example")
st.set.sim.pp("Maize", data.set = "maize.example")
loci <- vcf$ID
append.population("Combined", "Maize")
append.population("Combined", "Maize")
population.size.get("Combined")
```

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

Each derived cache is complete for all individuals or absent throughout a population. A mixed source cache causes all marker-genotype, genetic-value and phenotype caches to be cleared silently in the source and result.

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 individuals are retained; mixed source caches are cleared as described above. 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)

population.cross(name, pool1, pool2, n)
```

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.
`name`	Name of the offspring simulation population.
`pool1`, `pool2`	Names of two distinct, nonempty parental simulation populations. Neither may be named `"name"`.
`n`	Positive integer number of crosses.

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.

Crosses with a returned crossing design

Creates random crosses between two simulation populations and returns the selected parental identifiers at creation time.

Uses the existing diploid random-cross routine and its simulation random-number generator. Each offspring owns its full simulated pedigree independently of the returned design. The design records the immediate crossing keys; it is not reconstructed by parsing pedigree text. No genetic or phenotypic values are transferred. An existing offspring population is replaced, as by `cross`.

A data frame with `name` and two columns named after the parental populations. Rows follow offspring order and contain its stable identifier and the two selected parental identifiers.

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

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
reset.all()
st.set.sim.gp(data.set = "maize.example")
st.set.sim.mp(data.set = "maize.example")
st.set.sim.pp("Maize", data.set = "maize.example")
loci <- vcf$ID
cross("Offspring", "Maize", "Maize", 4)
population.size.get("Offspring")
```

data.params *Returns or inspects parameters associated with a data object*

Description

Returns or inspects 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 compatibility status of zero. The main effect is defining and loading the requested effects directly in memory.

`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

Locus names refer to the installed simulation map. Each generated effect map is named `EFF` followed by the locus name and `1`.

Value

NULL. The function is called for its side effect of defining effect maps for the locus names in the supplied data frame; existing matching effect maps are retained while other requested loci are processed.

Examples

```
data("v-tropmaize-map", package = "SelectionTools")
map <- v.tropmaize.map[1:12, c("chrom", "pos", "name")]
map$pos <- map$pos / 100
define.genome(map)
define.effects.df(map$name)
```

---

define.genome *Defines 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

```
data("v-tropmaize-map", package = "SelectionTools")
map <- v.tropmaize.map[1:12, c("chrom", "pos", "name")]
map$pos <- map$pos / 100
define.genome(map)
linkage.map.get()
lengths <- tapply(map$pos, map$chrom, max)
define.genome(map, length.vec = lengths)
```

---

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 Generates 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 Distributes 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 Distributes 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 converted and loaded through an in-memory stream using 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

```
data("v-tropmaize-map", package = "SelectionTools")
map <- v.tropmaize.map[1:12, c("chrom", "pos", "name")]
map$pos <- map$pos / 100
lengths <- tapply(map$pos, map$chrom, max)
set.genome.par(no.chrom = length(lengths), no.hom = 2, chrom.len = lengths)
description <- paste(map$name, "marker", paste0(map$chrom, "/", map$pos))
define.map(description)
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 compiled 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

Each derived cache is complete for all individuals or absent throughout a population. A mixed source cache causes all marker-genotype, genetic-value and phenotype caches to be cleared silently in the source and result.

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.

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

Description

Sets effect weights for all effects.

Usage

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

Arguments

weight	Weight value.

Value

An invisible list returned by the compiled 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 *Removes a stored effect map*

Description

Removes a stored effect map.

Usage

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

Arguments

Name	Name identifying the object to be processed.

Value

An invisible list returned by the compiled 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 *Removes all stored effect maps*

Description

Removes all stored effect maps.

Usage

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

Value

An invisible `list` returned by the compiled 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 *Evaluates all loci in a population*

Description

Evaluates 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

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
reset.all()
st.set.sim.gp(data.set = "maize.example")
st.set.sim.mp(data.set = "maize.example")
st.set.sim.pp("Maize", data.set = "maize.example")
loci <- vcf$ID
evaluate.allele("Maize", loci[1])
evaluate.allele("Maize", paste(loci[1:2], collapse = " "))
```

`evaluate.allele.freq` *Calculates or return allele frequencies for selected loci*

Description

Calculates 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

Vector-based input of loci is supported by `evaluate.allele.freq2`.

Examples

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
reset.all()
st.set.sim.gp(data.set = "maize.example")
st.set.sim.mp(data.set = "maize.example")
st.set.sim.pp("Maize", data.set = "maize.example")
loci <- vcf$ID
evaluate.allele.freq("Maize", loci[1])
evaluate.allele.freq("Maize", paste(loci[1:2], collapse = " "))
```

`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. Frequencies within one locus are obtained by evaluating 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

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
reset.all()
st.set.sim.gp(data.set = "maize.example")
st.set.sim.mp(data.set = "maize.example")
st.set.sim.pp("Maize", data.set = "maize.example")
loci <- vcf$ID
```

```
evaluate.allele.freq2("Maize", loci[1])
evaluate.allele.freq2("Maize", loci[1:2])
```

`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

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
reset.all()
st.set.sim.gp(data.set = "maize.example")
st.set.sim.mp(data.set = "maize.example")
st.set.sim.pp("Maize", data.set = "maize.example")
loci <- vcf$ID
evaluate.allele2("Maize", loci[1:2])
```

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

Further details appear in 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

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
reset.all()
st.set.sim.gp(data.set = "maize.example")
st.set.sim.mp(data.set = "maize.example")
st.set.sim.pp("Maize", data.set = "maize.example")
loci <- vcf$ID
evaluate.genotype("Maize", loci[1])
evaluate.genotype("Maize", paste(loci[1:2], collapse = " "), bg = 1)
```

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

Description

Calculates 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 *Calculates a haplotype statistic for a pair of loci*

Description

Calculates 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 *Releases or resets temporary haplotype-evaluation state*

Description

Releases or resets temporary haplotype-evaluation state.

Usage

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

Value

An invisible `list` returned by the compiled 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 *Initializes haplotype evaluation for a population*

Description

Initializes 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 compiled 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` *Returns genotype results from the current haplotype evaluation*

Description

Returns 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` *Returns the table produced by the current haplotype evaluation*

Description

Returns 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 *Evaluates haplotypes for selected loci in a population*

Description

Evaluates 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 *Evaluates linkage disequilibrium for a population and genomic position*

Description

Evaluates 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 *Evaluates selected loci in a population*

Description

Evaluates 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 *Evaluates marker-assisted donor-parent information for populations*

Description

Evaluates 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` *Evaluates 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 compiled 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 compiled 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

Calculates 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. Analysis of these dominant markers uses `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 *Constructs a matrix from genetic-distance results*

Description

Constructs 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. Analysis of these dominant markers uses `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

Returns status information concerning zero allele frequencies

Description

Returns 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 compiled 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

Further details appear in `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 compiled interface derives the population and marker dimensions and supplies the matrix together with allele and missing-value information to the 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

Further details appear in the `lib00.example1` for an application of the command.

---

genome.parameter.get *Returns 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 *Sets 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` *Extracts or summarizes genome segments for populations*

Description

Extracts or summarizes 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 compiled 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 *Returns the current marker-assisted donor-parent information*

Description

Returns 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 *Returns 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 *Returns stored information for individuals in a population*

Description

Returns 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 *Returns 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 *Builds estimation and validation subsets from a base data set*

Description

Builds 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.

Details

The two output sets must be distinct from each other and from the source. Both must contain at least one individual. Original allele codes, homologue order, phase state, linkage map and the selected phenotypic values are retained. Each output receives current marker statistics and automatically selected 4-, 8- or 16-bit allele storage. Both outputs are prepared and checked before either existing output is replaced.

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 *Builds training and prediction subsets from a base data set*

Description

Builds 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.

Details

The two output sets must be distinct from each other and from the source. Both must contain at least one individual. Original allele codes, homologue order, phase state, linkage map and the selected phenotypic values are retained. Each output receives current marker statistics and automatically selected 4-, 8- or 16-bit allele storage. Both outputs are prepared and checked before either existing output is replaced.

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 — *Builds the genomic-selection variance or covariance matrix for a data set*

Description

Builds 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 *Checks whether p-values are available for a genomic-selection data set*

Description

Checks 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 *Evaluates 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	Finite selected fraction in $(0, 1]$, used when N = G = 0. A value of 1 represents selection of all progeny.
N	Number of selected progeny for the finite-population correction. The counts are either both zero or finite positive whole numbers with N <= G. When both are positive, alpha is replaced by N/G.
G	Number of evaluated progeny for the finite-population correction. The same count restrictions apply as for N.

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

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

For selection of all progeny, `alpha = 1` or `N = G > 0`, the selection differential is zero and `es` equals the cross mean. An evaluated zero segregation variance is valid and also gives the cross mean. Both cross means and segregation variances must have been evaluated for the current data and marker effects. Invalid fractions, counts or unavailable prerequisites produce an error before the cross scores change.

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.

References

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

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.

See Also

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

`gs.cross.eval.gd` *Evaluates 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 *Evaluates crosses using genetic-distance function*

Description

Evaluates crosses using the 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 *Evaluates 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` *Evaluates 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 *Evaluates 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 *Evaluates 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	For "DH" and "SSD", the nonnegative integer number of additional random-intermating generations before line development. Zero corresponds to F1-derived DH lines or F2-derived SSD lines. For SSD, additional intermating starts from the F2; this is not the number of selfing generations. The parameter is method-specific for other 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"; "Kosambi" is also supported.

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; its existing calculation uses one generation irrespective of t.

For two inbred parents, pop.type = "SSD" describes the limiting variance among completely homozygous lines obtained by repeated selfing. For two loci with recombination frequency r, the parental-origin correlation in these lines is

$$\rho_{SSD}(r,t) = \frac{1-2r}{1+2r}(1-r)^t.$$

The corresponding probability of matching parental origin is $q = (1+\rho_{SSD})/2$. At `t = 0`, this gives $q = 1/(1+2r)$. Different chromosomes are independent. Finite selfing generations are represented by `ssd.mating`; they are not specified by increasing `t` in the analytical variance calculation.

The arguments are scalar values. Invalid population types, map functions or nonfinite, negative or fractional values of `t` produce an error before the stored cross scores change.

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.

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 *Information for Evaluated Crosses*

Description

Returns cross summaries with full numerical precision.

Usage

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

Arguments

`bestn`	Nonnegative whole number of entries to retain after sorting. A value of 0 returns all crosses. Values larger than the number of crosses return all available entries.
`sortby`	Character string naming the sorting criterion: `"index"`, `"gd"`, `"mu"`, `"mi"`, `"ma"`, `"va"` or `"es"`. The criterion must already have been evaluated unless `"index"` is used.
`out.filename`	Character string retained for compatibility. The result is returned directly from memory; this argument does not name an output file.
`data.set`	Name of the SelectionTools data set.

Details

The "index" order follows the parent indices. The "mi" criterion is sorted in increasing order; the other numerical criteria are sorted in decreasing order. Ties follow the parent indices.

Numeric results are returned directly from the stored calculations without rounding through formatted text. Display precision is controlled by R's printing options and does not alter the returned values.

Changing the marker data, map or marker effects invalidates affected cross summaries. Subsequent evaluations use the current data and effects. Invalid arguments and unavailable sorting criteria produce an error.

Value

Invisibly returns a data.frame containing the cross summaries. Columns P1Name and P2Name are character strings, so numeric-looking individual identifiers are retained unchanged. Columns P1No and P2No contain parent indices. The numeric columns gd, mu, mi, ma, va and es contain genetic distance, expected mean, lower and upper marker-effect limits, segregation variance and expected superior progeny value, respectively. Unevaluated criteria are NA; an evaluated zero remains zero. The segregation standard deviation is sqrt(va).

See Also

gs.cross.eval.mu, gs.cross.eval.va, gs.cross.eval.es

---

gs.cross.info.gd *Returns or writes genetic-distance information for crosses*

Description

Returns or writes 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. The result is returned directly from memory; this argument does not name an output file.
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. Cross-validation correlations are returned directly from memory.
out.filename.u	Character string retained for compatibility. Estimated marker effects are returned directly from memory.
auxfiles	Logical. If TRUE, both cross-validation result tables are returned directly from memory.
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 *Estimates marker effects using the LSQ method*

Description

Estimates 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 *Estimates marker effects using the RMLA method*

Description

Estimates 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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. The result is returned directly from memory; this argument does not name an output file.
`auxfiles`	Logical. If `TRUE`, the result is returned directly from memory.
`data.set`	character. Name of the data set that is used by the function

Details

The 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 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 *Estimates marker effects using the RMLR method*

Description

Estimates 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
data.set	character. Name of the data set that is used by the function.

Details

The 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 *Estimates 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 *Estimates genomic values for an estimation or prediction data set*

Description

Estimates 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.

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 *Returns the genomic-selection information matrix for a data set*

Description

Returns 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 *Returns the genomic-selection information level for a data set*

Description

Returns 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 *Returns the genomic-selection variance or covariance matrix for a data set*

Description

Returns 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 — *Returns the genomic-selection response vector for a data set*

Description

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

Usage

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

Arguments

data.set	Name of the SelectionTools data set.
names	Logical. A value of TRUE attaches individual identifiers as vector names.

Value

Invisibly returns the current numeric response vector y in internal individual order, including missing responses. With names = TRUE, each element is named with its individual identifier. The default returns an unnamed vector. Returns NULL if the response cannot be obtained or an argument is invalid.

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

Description

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

Usage

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

Arguments

data.set	Name of the SelectionTools data set.
names	Logical. A value of TRUE attaches individual identifiers as matrix row names.

Details

gs.make.A carries the row names of Z to both dimensions of the relationship matrix, so no separate name assignment is needed. The option changes only the returned matrix attributes; marker values, individual order and the internal data set are unchanged.

Value

Invisibly returns the current numeric marker/design `matrix` `Z` in internal individual order. With `names = TRUE`, row names are the individual identifiers, in the same order as `gs.get.y(data.set, names = TRUE)`. Column names are not added. The default returns an unnamed matrix. Returns `NULL` if the matrix cannot be obtained or an argument is invalid.

See Also

`gs.get.y`, `gs.make.A`

Examples

```
data("v-tropmaize-vcf", package = "SelectionTools")
st.load.vcf.data(v.tropmaize.vcf, data.set = "maize.names")
Z <- gs.get.Z("maize.names", names = TRUE)
A <- gs.make.A(Z)
head(rownames(Z))
identical(rownames(A), rownames(Z))
```

---

gs.info *Prints or handles a genomic-selection information message*

Description

Prints or handles 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 compiled 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 — *Estimates or sets genomic-selection lambda values using the aov method*

Description

Estimates or sets 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 — *Estimates or sets genomic-selection lambda values using the const method*

Description

Estimates or sets 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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` *Estimates or sets genomic-selection lambda values using the emstep method*

Description

Estimates or sets 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. The result is returned directly from memory; this argument does not name an output file.
`auxfiles`	Logical. If `TRUE`, the result is returned directly from memory.
`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` *Estimates or sets genomic-selection lambda values using the hsq method*

Description

Estimates or sets 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 — *Estimates or sets genomic-selection lambda values using the reg method*

Description

Estimates or sets 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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` *Estimates or sets genomic-selection lambda values using the rmla method*

Description

Estimates or sets 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. The result is returned directly from memory; this argument does not name an output file.
`auxfiles`	Logical. If `TRUE`, the result is returned directly from memory.
`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` *Estimates or sets genomic-selection lambda values using the rmla.02 method*

Description

Estimates or sets 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 — *Estimates or sets genomic-selection lambda values using the rmlc method*

Description

Estimates or sets 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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` *Estimates or sets genomic-selection lambda values using the rmlr method*

Description

Estimates or sets 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. The result is returned directly from memory; this argument does not name an output file.
`auxfiles`	Logical. If `TRUE`, the result is returned directly from memory.
`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` *Estimates or sets genomic-selection lambda values using the rmlv method*

Description

Estimates or sets 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 — *Estimates or sets genomic-selection lambda values using the rrblup method*

Description

Estimates or sets 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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.make.A *Constructs a genomic relationship matrix from marker dosages*

Description

Constructs the centred and scaled VanRaden relationship matrix from a matrix with individuals in rows and markers in columns.

Usage

```
gs.make.A(Z, coding = "0 1 2")
```

Arguments

Z	Nonempty numeric marker matrix without missing or infinite values.
coding	Marker coding: "0 1 2", "-1 0 1", or "0 0.5 1". Values must belong to the specified coding. "detect" requires exactly one compatible coding.

Details

After conversion to diploid allele dosage, frequencies are column means divided by two. With $W_{ij} = Z_{ij} - 2p_j$ and $c = 2\sum_j p_j(1 - p_j)$, the result is $A = WW^T/c$. The scale must be positive. Monomorphic columns contribute zero. The matrix can be singular; no inverse or diagonal adjustment is applied. Neither the input nor an internal SelectionTools data set is modified.

Value

A numeric square matrix in input individual order. If Z has row names, they label both dimensions. Returns NULL with a diagnostic for invalid input or a relationship-matrix construction failure.

gs.mme.coeff *Gs mme coeff*

Description

Builds or returns 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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

Builds or returns 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.

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

Builds or returns the penalized 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.

Value

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

---

gs.mme.restrhs *Gs mme restrhs*

Description

Builds or returns the model-specific 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.

Value

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

---

gs.mme.rhs *Builds or returns the right-hand side for genomic-selection mixed-model equations*

Description

Builds or returns 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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

Builds or returns 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 — *Builds or solves transformed mixed-model equations (coeff)*

Description

Builds or solves 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 — *Builds or solves transformed mixed-model equations (coeff.3)*

Description

Builds or solves 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 *Builds or solves transformed mixed-model equations (rhs)*

Description

Builds or solves 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 *Builds or solves transformed mixed-model equations (solve)*

Description

Builds or solves 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 *Sets or process all marker effects using the negative-effect convention*

Description

Sets 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 p column when p-values are available. Returns NULL if the required effect or map data are unavailable.

---

gs.plot.model.fit *Plots 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 *Sets or process all marker effects using the positive-effect convention*

Description

Sets 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.

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 — *Resets genomic-selection data and settings*

Description

Resets genomic-selection data and settings.

Usage

```
gs.reset()
```

Value

An invisible `list` returned by the compiled 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 — *Restricts marker data using the package genomic-selection interface*

Description

Restricts 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. The result is returned directly from memory; this argument does not name an output file.
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 *Returns marker p-values for a genomic-selection data set*

Description

Returns 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. The result is returned directly from memory; this argument does not name an output file.
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

Sets all info levels used by the genomic-selection routines

Description

Sets 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 compiled 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

Sets allele codes used by the genomic-selection routines

Description

Sets 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 *Sets effects used by the genomic-selection routines*

Description

Sets 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. The effects are transferred in memory.
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 *Sets info level used by the genomic-selection routines*

Description

Sets 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 compiled 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 *Sets lambda used by the genomic-selection routines*

Description

Sets 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. The result is returned directly from memory; this argument does not name an output file.
`auxfiles`	Logical. If `TRUE`, the result is returned directly from memory.
`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 *Sets num threads used by the genomic-selection routines*

Description

Sets 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 compiled 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`
Sets performance data used by the genomic-selection routines

Description

Sets performance data used by the genomic-selection routines.

Usage

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

Arguments

`y`	Numeric response vector in the order of the individuals in the data set. Its length must equal the number of individuals. Missing values (`NA` or `NaN`) can mark individuals without observations; infinite values are rejected.
`data.set`	Name of the SelectionTools data set.

Details

All individuals and marker data are retained. The stored vector can be retrieved with `gs.get.y`. The mixed-model interface `st.lmm` supports missing responses. Routines that require complete response vectors must be given only the observed values and corresponding inputs.

Value

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

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

Description

Performs 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 *Performs single-marker analysis for a genomic-selection data set*

Description

Performs 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 *Starts the genomic-selection timing helper*

Description

Starts the genomic-selection timing helper.

Usage

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

Arguments

`depth`	Timer depth, a finite whole number from 1 through 5.

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 *Stops the genomic-selection timing helper*

Description

Stops the genomic-selection timing helper.

Usage

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

Arguments

`depth`	Timer depth, a finite whole number from 1 through 5.
`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 *Tests marker or model parameters used by genomic-selection routines*

Description

Tests 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 compiled 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 *Tests estimated marker effects using the indicated method*

Description

Tests 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 *Tests estimated marker effects using the indicated method*

Description

Tests 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 *Tests estimated marker effects using the indicated method*

Description

Tests 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 *Estimates variance components for ridge-regression BLUP*

Description

Estimates 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	Main optimizer iteration budget. The shared engine may perform additional convergence polishing beyond this budget.
precision	Projected-gradient convergence tolerance in the shared mixed-model engine's scaled covariance parameters.
hsq	Heritability parameter.
out.filename	Character string retained for compatibility. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 — *Calculates weights or statistics used by the genomic-selection ANOVA method*

Description

Calculates 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 — *Writes 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. Further details appear in 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

Further details appear in init.population for an example

il.eval.library *Constructs or evaluates introgression-line libraries*

Description

Constructs or evaluates 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 *Constructs or evaluates introgression-line libraries*

Description

Constructs or evaluates 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 *Constructs or evaluates introgression-line libraries*

Description

Constructs or evaluates 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

Constructs or evaluates introgression-line libraries

Description

Constructs or evaluates 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.

`influence.st.lmm` *Exact Deletion Influence and Predictive Studentization*

Description

Refits a Gaussian mixed model after deleting cases or groups, preserving the calibrated fixed and random designs and covariance levels. Reports coefficient and covariance changes, Cook's distance, DFBETAs, predictive residuals, and the status of every requested deletion.

Usage

```
## S3 method for class 'st.lmm'
influence(model, group = NULL, units = NULL,
  control = list(), keep.fits = FALSE, ...)
## S3 method for class 'st.lmm'
rstudent(model, influence = NULL,
  control = list(), ...)
## S3 method for class 'st.lmm'
cooks.distance(model, influence = NULL, ...)
## S3 method for class 'st.lmm'
dfbeta(model, influence = NULL, ...)
## S3 method for class 'st.lmm'
dfbetas(model, influence = NULL, ...)
## S3 method for class 'st.lmm.influence'
summary(object, ...)
## S3 method for class 'st.lmm.influence'
print(x, ...)
## S3 method for class 'st.lmm.influence'
plot(x, which = c("cook", "dfbetas"),
  parm = NULL, labels = NULL, id.n = 3L, main = NULL, col = NULL, ...)
## S3 method for class 'st.lmm.influence'
cooks.distance(model, ...)
## S3 method for class 'st.lmm.influence'
dfbeta(model, ...)
## S3 method for class 'st.lmm.influence'
dfbetas(model, ...)
```

Arguments

`model`	For fitted-model methods, a converged `st.lmm` object fitted by ML or REML. The `rstudent` method also supports univariate method-of-moments models containing only fixed effects. For influence-object methods, an `st.lmm.influence` object.
`group`	NULL deletes one retained data row per case, including all its observed traits. A single character string names a column of the retained data whose values define groups. The alternative is a nonmissing atomic vector with exactly one value per

	retained row. A vector indexed to the original input rows is not accepted when its length differs from the retained data.
`units`	NULL requests every deletion unit. The alternative is a vector of distinct unit IDs or labels from the inventory. IDs follow the first appearance of each unit in the retained data. Units not requested remain visible in the result.
`control`	Named fitting controls that override the original controls for deletion fits. The original ML or REML method is retained.
`keep.fits`	Complete deletion models are retained when TRUE. Raw coefficient and covariance estimates, diagnostic status and warnings are retained regardless of this choice when a fit was returned.
`influence`	NULL computes deletion influence. The alternative is an influence result from the same fitted model. `rstudent` requires a complete case-deletion result; grouped or partial results are rejected.
`object`, `x`	An `st.lmm.influence` object.
`which`	Influence panels: `"cook"` and `"dfbetas"`, or their integer indices 1 and 2. Several distinct panels may be requested.
`parm`	NULL plots the maximum absolute DFBETA per deletion unit. Fixed-coefficient names or indices request a separate signed DFBETA panel for each selected coefficient.
`labels`	Optional character labels, one per inventory unit. The default uses the unit labels. Missing labels are not annotated.
`id.n`	Maximum number of units to annotate in each panel, selected by the largest absolute displayed influence. Zero suppresses annotations.
`main`	One title, or one title per requested panel.
`col`	One colour, or one colour per inventory unit.
`...`	For fitted-model `cooks.distance`, `dfbeta` and `dfbetas` without a supplied influence object, arguments to `influence`. For plotting, graphical parameters such as `xlim`, `ylim`, `xlab`, `ylab`, `pch` and `cex`. For printing, arguments to the result-table printer. Other methods currently ignore additional arguments.

Details

Repeated fits run in the current R process. Numerical kernels use the existing OpenMP and BLAS thread controls; `st.set.num.threads` describes these controls. Intermediate state stays in C-owned memory.

Deletion fitting: Deletion marks the requested observed response cells missing in the compiled model. The full fixed design, random and residual coefficient bases, factor levels, known relationships and covariance kernels remain calibrated to the original fit. Predictor transformations are not reevaluated. Every free covariance parameter is re-estimated by the original ML or REML method. Fixed covariance blocks retain their supplied values.

A disappearing grouping level remains available for prediction, including its relationships to observed levels in a known covariance matrix. Deleting all observed support for a free covariance stratum instead gives an explicit `"lost covariance stratum"` result. The function does

not silently remove or fix that covariance parameter. With `homdiag` or homogeneous compound-symmetric or AR1 residual covariance, an occasion may lose all its observations if other occasions still inform the common scale. The fitted occasion positions and AR1 gaps are retained. Losing all support for a separate occasion variance or an entire common-scale stratum remains a lost-stratum failure.

Every requested unit retains its warnings, convergence and rank information. Failed fits, nonconvergence, nonestimable held-out means, changed fixed rank, zero prediction-error variance and insufficient observations are reported explicitly. A failure does not stop unrelated deletion units. Failed or undefined inferential diagnostics remain `NA`; raw estimates can still be inspected when the fitting engine returned them.

Coefficient changes: Changes use the original estimate minus the deletion estimate. `dfbeta` returns fixed-coefficient changes, only for individual coefficients estimable in both fits. `dfbetas` divides each such change by that coefficient's deleted-fit standard error. These standard errors use the unadjusted fitted covariance, even if the original model requested Kenward–Roger inference.

Cook's distance is the squared change in the estimable fixed-coefficient vector, measured with the inverse original unadjusted fixed-coefficient covariance, divided by the original fixed rank. Deleted coefficients are mapped into the original independent fixed basis before this calculation. Cook's distance is undefined if the estimable fixed-effect space changes, the deletion does not converge, or the original fixed rank is zero. This convention is stated in each result and does not use the deleted-fit covariance in the Cook quadratic form.

Full random and residual coefficient covariance matrices and their changes are retained in their original coefficient units. Unidentified covariance coordinates should be assessed through implied covariance and recorded boundary diagnostics, rather than interpreted as precisely estimated individual parameters.

Predictive studentization: The deleted fit predicts the held-out response using its full Gaussian covariance, including random-effect and residual cross-covariance. The indices o and t identify the remaining observed and deleted cells, respectively. The cross-covariance is $K = V_{to}$, and $C = (X_o' V_{oo}^{-1} X_o)^{-1}$ is defined on an independent fixed basis. The predictor, before restoring known offsets, is $X_t\hat{\beta} + KV_{oo}^{-1}(y_o - X_o\hat{\beta})$. With $D = X_t - KV_{oo}^{-1}X_o$, its prediction-error covariance is $V_{tt} - KV_{oo}^{-1}K' + DCD'$.

`rstudent` divides the observed-minus-predicted response by the square root of the corresponding diagonal of this covariance. All observed traits of a case are deleted together. The covariance parameters are fixed at their deletion estimates for this calculation. No exact t reference distribution is asserted for general mixed models. For ordinary Gaussian regression fitted by REML this agrees with conventional externally studentized residuals.

For a univariate fixed-only method-of-moments model, `rstudent` uses the analytic externally studentized regression residual. Positive precision weights multiply the response residual by their square root; the deleted residual variance uses the weighted residual sum of squares and DF minus one, where DF is the number of observations minus the fixed-design rank. This method requires `influence=NULL` and an empty `control` list; supplied influence objects and deletion-fit controls are rejected.

Plots: The default panels show Cook's distance and the maximum absolute DFBETA for each unit. `parm` selects individual signed DFBETA panels. Unavailable and unrequested units retain their status and are counted in the plot footer; no zero influence is substituted. If a panel has no finite values it displays that fact explicitly. Plots annotate the largest displayed values and

invisibly return their exact points and annotations. They do not alter the graphics layout or assign a scientific explanation to an influential case or group.

Value

`influence` returns class `st.lmm.influence`, containing an all-unit `summary`, `inventory`, per-unit `results`, `requested` IDs, `fixed.change`, `dfbetas`, `Cook`, fitting controls, method and diagnostic definitions. The inventory maps retained rows to stable unit IDs. Each result records row and response-cell keys, status, warnings, fit diagnostics, lost strata, estimates, changes, and a held-out prediction table and full prediction-error covariance when available. Complete deletion fits are included only with `keep.fits=TRUE`.

`summary` returns the all-unit table. `cooks.distance` returns a named vector; `dfbeta` and `dfbetas` return matrices with one row per inventory unit and one column per original fixed coefficient. `rstudent` returns a matrix with one row per observation and one column per trait, restores omitted selected rows under `na.exclude`. For likelihood fits it attaches the deletion-status table. Originally missing responses and unavailable deletion diagnostics remain `NA`.

See Also

`st.lmm`, `st.lmm.diagnostics`, `st.lmm.refit`, `st.lmm.components`

`info` *Prints or handles an information message*

Description

Prints or handles an information message.

Usage

```
info(lev, msg)
```

Arguments

`lev`	Information or verbosity level.
`msg`	Message text.

Value

An invisible `list` returned by the compiled 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 *Information-level-controlled status output*

Description

A status message is signaled 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

The function returns `NULL` invisibly. Message text is evaluated only when the requested information level is enabled.

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 numeric columns named `ind`, `chrom`, `hom`, `pos`, and `all`, representing individual, chromosome, homologue, segment position, and allele.
`delete`	Logical. `TRUE` replaces an existing population with the same name after the input data have passed validation.

Details

The input data frame has exactly the five named numeric columns described above. Individual, chromosome and homologue numbers and allele codes are finite whole numbers; positions are finite and lie within the defined genome. 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.

Malformed input is rejected before an existing population is changed. Once validation succeeds, `delete=TRUE` removes an existing population of the same name 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

`NULL`, invisibly. The function initializes the named simulation population from the supplied chromosome-segment data.

Examples

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
reset.all()
st.set.sim.gp(data.set = "maize.example")
st.set.sim.mp(data.set = "maize.example")
st.set.sim.pp("Maize", data.set = "maize.example")
loci <- vcf$ID
observed.population <- get.population("Maize")
init.population("Imported", observed.population)
evaluate.genotype("Imported", loci[1])
```

`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` *Calculates or summarize linkage drag*

Description

Calculates 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` *Returns a linkage map used by the simulation routines*

Description

Returns the currently active simulation linkage map.

Usage

```
linkage.map.get()

population.marker.map(n.markers)

extend.marker.map(n.markers)
```

Arguments

`n.markers`	Total number of map points after the operation. For replacement, at least the number of chromosomes in the defined genome; for extension, at least the current map size.

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.

Random simulation marker map

Replaces the simulation marker map with a specified number of random marker positions.

Places at least one marker on each chromosome and allocates remaining markers with probabilities proportional to chromosome length. Positions are uniform within chromosomes and sorted physically. Marker names are `M1`, `M2`, and so on. Uses the R random-number generator; `set.seed` provides reproducibility. The chromosome segments and stored pedigrees of existing simulation populations are retained. Genotypes at the new positions are obtained when populations are exported to marker datasets.

The active linkage map, invisibly.

Extension of an existing marker map

`extend.marker.map` retains every existing locus with its identifier, position and class, and adds random marker positions until the map contains `n.markers` points in total. New positions are uniform over genome length; new identifiers avoid existing names. The result is sorted physically within chromosomes. Chromosome lengths, inherited segments and stored pedigrees are preserved. Genotypes at the new loci are derived from those inherited segments, not additional observed founder genotypes. The operation invalidates cached simulation marker genotypes, which are rebuilt when requested. As with map replacement, simulation effect-map definitions and cached evaluated values are cleared; this operation does not change marker-dataset model results.

An equal count is a no-op and does not advance the random-number generator. A smaller count is rejected without changing the map. Uses the R random-number generator; its seed determines reproducibility. Returns the active map invisibly.

`linkage.map.load` *Loads 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` *Saves 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

Further details appear in `define.effects` for an example

list.populations *list.populations*

Description

Returns the names and active sizes of the currently supported 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 compiled 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

Further details appear in define.effects for an example

load.internal.effmap *Loads an internal effect map*

Description

Loads 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 compiled 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` *Runs and Compares 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. Further details appear in 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. Scenario 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(). Further details appear in 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.

A NULL result from any simulation aborts the comparison before later scenarios are run. Files from completed scenarios remain intact; the operation does not roll back the whole comparison.

Value

On successful completion, NULL is 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`	Requested simulation-name identifier. It must match the identifier stored in the result; it does not determine the file name.
`simulation.run`	Requested simulation-run identifier. It must match the identifier stored in the result; 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`.

Details

The stored `simulation.name` and `simulation.run` must match the requested identifiers. The result is loaded through an isolated environment, so other variables saved in the file cannot overwrite the requested identifiers or other caller bindings.

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, for each individual, linkage drag is averaged over donor-bearing homologues at that target; it is zero when neither homologue carries the donor allele.

SM Number of marker data points attributed to single-marker assays.

HT Number of marker data points attributed to high-throughput assays.

Further details appear in 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. Further details appear in `recurrent.parent`.

`linkage.map`
: Name of the file, in which the linkage map is described. Further details appear in 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. Each doubled-haploid generation uses its own corresponding entry. 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. Zero selects the chromosome end; together with a zero reg.begin, it specifies the whole chromosome. 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.

Details

With gen.type=NULL, input recoding is available without a simulated generation sequence. Every supplied entry is one of "f1", "bc", "s" or "dh"; unsupported generation types are rejected. In the first generation, "f1" and "bc" cross the donor and recurrent parents, "s" selfs the donor parent, and "dh" produces doubled haploids from the donor parent. In later generations, "f1" again crosses the supplied parents, "bc" crosses the previous selected population with the recurrent parent, and "s" and "dh" use the previous selected population.

Every supplied target, flanking or recipient locus must occur on the linkage map. A list containing an unknown locus is rejected even when other supplied loci are present.

Regional bounds are finite nonnegative distances in cM from the telomere. The beginning is strictly smaller than the effective end, and the end lies within the simulation chromosome. A zero end denotes the chromosome end.

Simulation work uses a separate state scope. Caller populations, genome, linkage map, effects, individual identities and marker data point state are preserved on success and failure. Random draws intentionally advance the simulation RNG. Shared storage capacity and allocation diagnostics can change; this state preservation does not imply whole-operation rollback.

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. For each individual, linkage drag is averaged over the homologues carrying the donor allele at that target, and is zero when neither homologue carries it.

`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.

Invalid requests either return `NULL` invisibly or signal an error, depending on the condition. Only successful simulations save a new result in the explicitly supplied `result.file`. Failures before saving leave any previously successful result file intact.

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. A value of `TRUE` includes a summary of the main simulation parameters.

`ext.summary`
: Logical. A value of `TRUE` includes an extended summary of the simulation parameters.

`RPG.gw.mean`
: Logical. A value of `TRUE` includes mean genome-wide recurrent-parent genome (RPG) values.

`TargetAlleles`
: Logical. A value of `TRUE` includes the mean target-allele summary.

`RPG.gw.Q10`
: Logical. A value of `TRUE` includes the Q10 value of the genome-wide RPG.

`RPG.reg.Q10`
: Logical. A value of `TRUE` includes Q10 values of RPG in the defined chromosome regions.

`RPG.reg.mean`
: Logical. A value of `TRUE` includes mean RPG values in the defined chromosome regions.

`DonorSegments.gw`
: Logical. A value of `TRUE` includes the mean number of donor segments genome-wide.

`DonorSegments.reg`
: Logical. A value of `TRUE` includes the mean number of donor segments in the defined chromosome regions.

`LinkageDrag`
: Logical. A value of `TRUE` includes mean linkage-drag summaries for the target loci. Individual values average over donor-bearing homologues at each target and are zero when neither homologue carries the donor allele.

`MarkerAnalyses`
: Logical. A value of `TRUE` includes 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 character vectors in one R data object for the file-based `mab.*` interface.

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 file-based marker-assisted backcrossing interface identifies these input files by their list-element names.

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`

optimize.population *Optimizes storage for simulation populations*

Description

Compacts segment storage for the requested simulation populations while preserving all stored segments.

Usage

```
optimize.population(PopNames)
```

Arguments

PopNames	Population name or population specification.

Details

Segment storage is compacted for the actual individuals in each requested population. Unused individual slots are released and the individual array is trimmed to the population size. Chromosomes, individual identities and complete derived caches are preserved. Segment capacity follows the reservation set by `set.NoLociInit`.

Completely free backing blocks are released from the shared simulation memory pool after successful optimization. Blocks containing any live allocation are retained. Other populations' live allocations are not moved or released. Released memory need not immediately reduce the process's resident memory on every system. Later simulation growth can allocate storage again.

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 *Creates a linkage map for phenotype or simulation use*

Description

Creates 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` *Generates 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, $V_{G,i}$ denotes the sample variance of the effect-specific genetic values and h_i^2 denotes 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 *Performs the SelectionTools operation "plabsim"*

Description

Performs 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 *Initializes the PLABSIM simulation state*

Description

Initializes 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 *Performs the SelectionTools operation "plabsim R date"*

Description

Performs 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 *Performs the SelectionTools operation "plabsim R version"*

Description

Performs 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 *Performs the SelectionTools operation "plabsim version"*

Description

Performs 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 *Creates or process a plant object in the simulation framework*

Description

Creates 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 *Appends 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. Its individuals are retained; mixed derived caches are cleared silently.

Details

Raw chromosome data are copied for every appended individual. Cached marker genotypes, genetic values and phenotypes are cleared throughout the resulting destination population, including when it was initially empty. The source population is retained. A mixed source or destination cache causes all three caches in that population to be cleared silently.

Value

An invisible list returned by the compiled 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 compiled 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 *Copies 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

Each derived cache is complete for all individuals or absent throughout a population. A mixed source cache causes all marker-genotype, genetic-value and phenotype caches to be cleared silently in the source and result.

The source individuals are retained. 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 compiled 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 *Divides 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

Each derived cache is complete for all individuals or absent throughout a population. A mixed source cache causes all marker-genotype, genetic-value and phenotype caches to be cleared silently in the source and result.

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 compiled 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 *Checks 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 *Removes one individual from a population*

Description

Removes one individual from a population.

Usage

```
population.individual.remove(PopName, IndPos)
```

Arguments

PopName	Name of the population.
IndPos	One whole-number individual position. An out-of-range position leaves the population unchanged.

Value

NULL, invisibly. A valid individual position removes that individual while preserving the order of the remaining individuals. An out-of-range position leaves the population unchanged.

population.info.get *Returns stored information for an individual*

Description

Returns the character information stored for an individual in a simulation population.

Usage

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

population.pedigree(name, source = c("auto", "simulation", "marker"))
```

Arguments

PopName	Name of the population.
ind	Individual index or indices.
name	One simulation-population or marker-dataset name.
source	Registry to query. The default requires an unambiguous existing name.

Value

A character value containing the stored information for the requested individual when the individual index is valid; otherwise NULL.

Whole-population simulated pedigree

Returns stable individual names and their complete stored simulated pedigree expressions from a simulation population or marker dataset.

Imported founders retain their exact imported identifiers. New crosses combine the stored pedigree expressions of the actual parents as (left_x_right); DH and SSD production append the respective operation inside parentheses. The complete expression is stored when the individual is created and remains available after ancestor populations are removed. Copying, moving, sorting, selection and population renaming retain each individual's identity. This accessor does not infer a pair of immediate parents from a complex pedigree.

The result follows current active individual order. Repeated copies in a simulation population may have the same identity. Export to a marker dataset rejects duplicated individual identifiers. A missing query does not create a marker dataset. A name present in both registries requires an explicit source.

A fresh data frame with character columns name and pedigree, using automatic row names. Empty valid populations return zero rows.

---

population.info.set *Sets 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 supported 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 supported 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 compiled 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` *Optimizes storage for simulation populations*

Description

Compacts segment storage for the requested simulation populations while preserving all stored segments.

Usage

```
population.optimize(PopNames)
```

Arguments

`PopNames`	Population name or population specification.

Details

Segment storage is compacted for the actual individuals in each requested population. Unused individual slots are released and the individual array is trimmed to the population size. Chromosomes, individual identities and complete derived caches are preserved. Segment capacity follows the reservation set by `set.NoLociInit`.

Completely free backing blocks are released from the shared simulation memory pool after successful optimization. Blocks containing any live allocation are retained. Other populations' live allocations are not moved or released. Released memory need not immediately reduce the process's resident memory on every system. Later simulation growth can allocate storage again.

Value

An invisible `list` returned by the compiled 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`
Loads 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 *Saves 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 *Removes 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 returned to the package's compiled memory pool. Other simulation-wide definitions such as the genome, linkage map, and effects are separate and are not removed by this operation.

The pool retains backing memory for subsequent simulation populations. Its default backing-block increment is 100 MB (100000000 bytes); a larger contiguous request can require a multiple of that increment. Population deletion therefore need not reduce the process's resident memory. The pool's backing memory is released when the package's compiled library is unloaded.

Value

An invisible `list` returned by the compiled 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` *Removes all simulation populations*

Description

Removes all currently supported 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.

Population storage is returned to the package's compiled memory pool. Backing blocks remain available for subsequent simulations, as described for `population.remove`.

Value

The function returns `NULL` invisibly after the population-removal request.

population.rename *Rename a simulation population*

Description

Renames a supported 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 compiled 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 compiled 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` *Samples 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

Each derived cache is complete for all individuals or absent throughout a population. A mixed source cache causes all marker-genotype, genetic-value and phenotype caches to be cleared silently in the source and result.

The source individuals are retained. 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 compiled 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 *Returns 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 *Sorts 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. TRUE removes the source after transfer by concatenation; FALSE retains the source and appends 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.

propsd *Proportional Loss of Expected Sample Standard Deviation*

Description

Evaluates the proportional loss of expected sample standard deviation under a fixed Gaussian covariance model. Exact sample-variance moments determine a two-moment Gamma approximation to the expected sample standard deviation.

Usage

```
propsd(G, parents = NULL, counts = NULL)

## S3 method for class 'propsd'
print(x, digits = NULL, ...)
```

Arguments

G	Complete finite symmetric positive-semidefinite covariance-proportional matrix with at least two reference parents and positive centered contrast variance. Singular matrices are supported. Both dimensions use the same order. Row and column names, when present, are complete, unique and match in order. The reference contains one occurrence per row. Its covariance scale and allele-frequency reference remain fixed.
parents	Selected row indices or parental names, with repetitions permitted. Repeated occurrences represent the same parental random variable. At least two selected occurrences are required. Cannot be combined with counts. Character labels for an unnamed matrix are "1", "2", and so on.
counts	Alternative nonnegative integer multiplicities in full reference order. Optional names match that order exactly. Each count and their sum must be at most 2^53 - 1; the sum is at least two. Fractional gene contributions are not multiplicities.
x	A propsd result.
digits	Optional printing precision, otherwise following st.options.
...	Additional printing arguments.

Details

With both selected-sample arguments NULL, the selected sample equals the complete reference. For a covariance matrix B describing n sample occurrences, define $H = I - 11'/n$ and $C = HBH$. Under a zero-mean Gaussian model the ordinary sample variance has exact moments

$$\mu_B = \operatorname{tr}(C)/(n-1), \qquad v_B = 2\sum_{ij} C_{ij}^2/(n-1)^2.$$

Moment matching gives Gamma shape $a = \mu_B^2/v_B$ and scale $b = v_B/\mu_B$. The expected standard deviation is approximated by

$$e(B) = \sqrt{b}\,\Gamma(a+1/2)/\Gamma(a).$$

For the selected multiset s, its covariance is the indexed matrix $G_s = G[s,s]$, including duplicate rows and columns. The returned loss is

$$1 - e(G_s)/e(G).$$

The sample-variance moments are exact under the model; the expected-SD formula is generally an approximation. It is exact when the nonzero centered covariance eigenvalues are equal. It is not the standard deviation of fitted parental merits, group relatedness, or a mean of simulated SD ratios.

Multiplicities are evaluated without expanding a replicated matrix. For $m = \sum_i k_i$, $w = k/m$, $r = Gw$, $q = w'Gw$, and $D_{ij} = G_{ij} - r_i - r_j + q$, the selected moments are

$$\mu_s = \frac{\sum_i k_i D_{ii}}{m-1}, \qquad v_s = \frac{2\sum_{ij} k_i k_j D_{ij}^2}{(m-1)^2}.$$

An arbitrary scaling of all multiplicities can change the result because ordinary sample variance uses $m - 1$. The occurrence convention is therefore explicit.

A common positive covariance scale cancels from the loss. A common random-intercept covariance cancels under centering. The implementation checks symmetry and positive semidefiniteness, removes common intercept terms using anchored contrasts, and scales covariance contrasts before evaluating moments with extended-precision accumulators. Each selected active sample is anchored and scaled directly from the original matrix; unused high-variance reference parents therefore do not erase smaller selected contrasts. It uses log Gamma arithmetic; for shape above 128 the log ratio correction uses $-1/(8a) + 1/(192a^3) - 1/(640a^5)$. No replicated covariance workspace is made. Reported moments use the original covariance units and can overflow or underflow the R numeric range at extreme scales even when the loss and expected SD remain well defined.

A zero-contrast reference is undefined and produces an error. A selected sample with at least two identical parental occurrences has zero expected SD and loss one. Negative losses are valid and are retained without clipping. Input matrices, random-number states and thread settings are preserved.

Value

A list of class `propsd`, containing:

`loss`	Computed proportional loss, possibly negative.
`expected.sd`	Named vector `reference`, `selected` of Gamma-approximated expected sample standard deviations in the square-root units of the supplied covariance matrix.
`n`	Named vector of reference and selected occurrence counts.
`counts`	Selected multiplicities in complete reference order.
`moments`	Data frame with rows `reference` and `selected`; columns `mean.variance`, `variance.variance`, `gamma.shape` and `gamma.scale`. Gamma parameters are undefined and reported as `NA` for a zero-contrast selected sample.
`method`	Qualification identifying Gaussian sample-variance moments and the two-moment Gamma approximation.

References

Fern\'andez-Gonz\'alez, J., Metwally, S. M., and Isidro y S\'anchez, J. (2026). MateR: a novel genomic mating framework. *Genetics* 232, iyag013. doi:10.1093/genetics/iyag013. Equations 14–15 and Appendix C.

See Also

`st.cross.plan`, `st.get.A`, `st.options`

`read.vcf` *Reads 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", pedigree = TRUE,
         strictACTG = FALSE)
```

Arguments

pedigree	Logical; controls writing or reading quoted SelectionToolsPedigree records in the VCF header.
strictACTG	Logical, default FALSE. When enabled, the reader accepts only A, C, G and T allele labels or their abbreviations Ade, Cyt, Gua and Thy, and uses fixed internal codes A/Ade=1, C/Cyt=2, G/Gua=3 and T/Thy=4, independent of REF/ALT order. Matching is case-insensitive. Other allele labels are treated as missing.
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 compiled parser is called.
data.set	Character string naming the SelectionTools marker data set to create or replace.

Details

Full simulated pedigrees are stored in quoted SelectionToolsPedigree header records within the VCF, including compressed files. Sample names retain stable individual identifiers and may contain long pedigree-based names. Without these records, imported individuals are founders whose pedigree equals their sample name. Incomplete, duplicate or inconsistent pedigree records cause import to fail before replacing the destination. pedigree=FALSE omits these records on output and ignores them on input. Fitted prediction state is retained by the in-memory STvcf representation, rather than plain VCF.

The compiled 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.

With strictACTG=TRUE, labels in REF and ALT determine the fixed nucleotide codes instead. The option retains a marker if at least one of its REF/ALT labels is A, C, G, T, Ade, Cyt, Gua or Thy (case-insensitive). It discards markers with no such label; an accepted label need not be observed in a nonmissing genotype. Indels, symbolic alleles, spanning-deletion stars, ambiguous bases and numeric allele labels are not accepted as nucleotides. If either allele of a diploid call is excluded, the entire call is missing. Genotype order, retained marker identifiers, map positions and individual pedigrees are preserved; chromosome codes are consecutive after filtering. If no marker remains, import fails without replacing an existing destination. The default FALSE preserves the original import behavior.

Fixed nucleotide coding aligns matching marker identifiers with different REF/ALT orders. It does not match marker identifiers or positions, complement opposite strands, or normalize indel representations.

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 supported 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

Vector-based input of loci is supported by `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` *Removes 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 compiled 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

Removes 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 compiled 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

Removes 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 compiled 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 supported 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()
```

Details

Storage owned by simulation populations is returned to the package's compiled memory pool. The pool retains backing blocks for reuse by subsequent simulations. Further details appear in `population.remove` for the storage lifecycle.

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 *Resets marker-assisted donor-parent information*

Description

Resets marker-assisted donor-parent information.

Usage

```
reset.mdp()
```

Value

An invisible `list` returned by the compiled 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 — *Returns or displays resource information used by SelectionTools*

Description

Returns or displays 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` *Chooses the simulation random-number generator state*

Description

Chooses 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 compiled 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 compiled 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` *Initializes the random-number generators*

Description

Initializes 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. Thread configuration precedes initialization, and OpenMP settings remain constant during the run. Normal top-level execution with up to 256 threads is supported; shared fallback execution is outside this guarantee. A repeated complete experiment requires a separate reset of simulation data; `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 compiled routine. The function is primarily called for its initialization side effects.

Examples

```
st.set.num.threads(1)
rng.init(seed = 1001)
# A simulation runs after preparing its input data.
rng.init()
```

rng.list *List the simulation random-number generator state*

Description

Lists 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 compiled 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

Each derived cache is complete for all individuals or absent throughout a population. A mixed source cache causes all marker-genotype, genetic-value and phenotype caches to be cleared silently in the source and result.

Sampling retains the source individuals; mixed source caches are cleared as described above. 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

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
reset.all()
st.set.sim.gp(data.set = "maize.example")
st.set.sim.mp(data.set = "maize.example")
st.set.sim.pp("Maize", data.set = "maize.example")
loci <- vcf$ID
sample.population("Sample", "Maize", NoI = 4, rep = 0)
population.size.get("Sample")
```

---

save.linkage.map *Saves a linkage map*

Description

Saves 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 *Calculates a standard-deviation quantity used by the simulation code*

Description

Calculates 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 *Defines a selection target*

Description

Defines 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` *Selects 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` *Determines 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` *Selects 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

Each derived cache is complete for all individuals or absent throughout a population. A mixed source cache causes all marker-genotype, genetic-value and phenotype caches to be cleared silently in the source and result.

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

`NULL`, invisibly. The function constructs `newPop` from the individuals matching the requested allele criteria.

See Also

[select.all.best()], [select.n.best()], [define.effects()]

Examples

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
reset.all()
st.set.sim.gp(data.set = "maize.example")
st.set.sim.mp(data.set = "maize.example")
st.set.sim.pp("Maize", data.set = "maize.example")
loci <- vcf$ID
observed <- evaluate.genotype("Maize", loci[1], bg = 1)
criteria <- data.frame(locus = loci[1],
                       allele1 = as.integer(observed[[1]][1]),
                       allele2 = as.integer(observed[[2]][1]))
select.genotypes("Selected", "Maize", criteria)
evaluate.genotype("Selected", loci[1])
```

---

select.n.best *Selects 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

Each derived cache is complete for all individuals or absent throughout a population. A mixed source cache causes all marker-genotype, genetic-value and phenotype caches to be cleared silently in the source and result.

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`

Selects the best genome segments according to the requested criterion

---

Description

Selects 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.

Details

Each derived cache is complete for all individuals or absent throughout a population. A mixed source cache causes all marker-genotype, genetic-value and phenotype caches to be cleared silently in the source and result.

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 *Sets crossover-frequency information*

Description

Sets crossover-frequency information.

Usage

```
set.co.freq(cofreq=1)
```

Arguments

`cofreq`	Function argument used by this operation.

Value

An invisible `list` returned by the compiled 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 compiled 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.
Further details appear in `define.effects` 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 compiled 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` *Sets marker-assisted donor-parent information*

Description

Sets 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 compiled 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` *Sets the initial segment capacity used by the simulation*

Description

Sets the initial and minimum segment-storage reservation per homologue for simulation individuals.

Usage

```
set.NoLociInit(NoLociInit)
```

Arguments

`NoLociInit`	Positive initial segment capacity per homologue. The default is 10.

Details

Capacity grows geometrically when more segments are needed. Simulation operations reserve predictable storage in advance and reuse existing capacity on subsequent calls. This setting controls storage reservation; the stored segments are determined by the simulation operations.

`population.optimize` can compact the storage of selected populations using this reservation as the rounding unit.

Value

An invisible `list` returned by the compiled 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 segment capacity.

`set.population.gvalue` *Sets 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 *Sets population information used by the simulation routines*

Description

Sets 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`	Dry-run diagnostics from `st.phase(commit = FALSE)`, `NULL` after a committed call.
`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 *Creates or evaluate a single cross*

Description

Creates 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 *Starts the simulation timing helper*

Description

Starts the simulation timing helper.

Usage

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

Arguments

`depth`	Timer depth, a finite whole number from 1 through 5.

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 *Stops the simulation timing helper*

Description

Stops the simulation timing helper.

Usage

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

Arguments

`depth`	Timer depth, a finite whole number from 1 through 5.
`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 compiled 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 *Calculates 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", "Dp-estimate-phases", and "r-estimate-phases". The last option returns the signed gametic correlation after phase estimation.
auxfiles	Logical. If TRUE, the pairwise results are returned directly from memory 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. Chromosome groups and marker order are obtained from the data set's map.

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, f_x and f_y denote the allele frequencies and f_{xy} denotes 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"`, `"Dp-estimate-phases"`, and `"r-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. The quantities 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, r^2, or D' is calculated from the estimated haplotype frequencies. The estimates maximize the observed-data likelihood for the two-locus phase-ambiguity problem; Clayton and Leung (2007) discuss haplotype estimation for association data.

For `"r-estimate-phases"`, the signed statistic is

$$r = \frac{h_{00} - p_0 q_0}{\sqrt{p_0(1-p_0)q_0(1-q_0)}}.$$

Here allele zero is the first biological allele in the marker's sorted internal allele-code order, and p_0 and q_0 are its two marginal frequencies among complete pairs. Positive values indicate coupling of these two alleles; exchanging the allele labels at one locus reverses the sign. The statistic is a gametic correlation estimated from the two-locus likelihood, not a correlation of observed diploid allele dosages. Reversing the order of the two alleles within an individual's unphased genotype has no effect.

Signed r is missing if no complete genotype pairs remain, a locus becomes monomorphic among complete pairs, the EM calculation does not yield a converged solution, or equally supported likelihood maxima have opposite signs. The last case can occur when all complete observations are double heterozygotes; their phase does not identify a sign. The existing r^2 and D' phase-estimation paths retain their zero value for a pair with no usable variation or no converged solution.

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 result-transfer time for large data sets.

A general review of LD measures and their dependence on allele frequencies is given by 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`. `Chrom` is the one-based chromosome index in the selected data set. `Locus1` and `Locus2` are one-based marker positions within that 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.

st.calc.ld.2 *Calculates linkage disequilibrium from marker data*

Description

Calculates 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 availability of the LD result. With keep.ldfile = FALSE, the result table is returned directly from memory.
keep.ldfile	Logical. If FALSE, the LD result is returned to R directly from memory. 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 LD result is requested in memory. 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 *Calculates the package-specific Q statistic or quantity*

Description

Calculates 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 result is returned directly from memory.
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 *Calculates recombination fractions*

Description

Calculates 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 result is returned directly from memory.
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 *Calculates Haplotype Enrichment and Selects 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. Exactly one of haplotype.options and blocks is required.
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 compiled 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. compiled 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 *Returns chromosome statistics for a data set*

Description

Returns 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. Results are returned from memory without creating a file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 *Compacts a Marker Data Set and Repacks Allele Storage*

Description

Releases calculated results and reusable workspace allocations associated with one SelectionTools marker data set while retaining its marker data and linkage map. st.repack.data.set() selects compact allele storage while retaining the data set and calculated results.

Usage

```
st.compact.data.set(data.set = "default")

st.repack.data.set(data.set = "default", commit = TRUE)
```

Arguments

data.set	Character string naming the marker data set.
commit	Logical. For repacking, TRUE replaces storage; FALSE verifies and reports the proposed representation without changing the data set.

Details

The retained core consists only of individual and marker names, the allele storage 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.

st.repack.data.set() retains calculated state and chooses the smallest supported representation from the maximum number of observed nonmissing alleles at any marker: at most three gives four bits per diploid genotype, four to fifteen gives eight bits, and sixteen or more gives sixteen bits. Four- and eight-bit storage includes a per-marker dictionary that preserves original allele codes, missing values, and homologue order. Dictionary memory is additional to the genotype payload. Imports and genotype replacements select this storage automatically. Repacking is useful after changes that reduce allele diversity. New storage is fully constructed and checked before replacing live storage.

Values that are required again must be recalculated or reloaded by the corresponding SelectionTools function. The dedicated genomic_relationship_A cache and dependent selected hybrid SCA caches are cleared. Hybrid performance and parental references remain supported, and requested relationship matrices are rebuilt on demand.

Value

Invisibly returns the estimated amount of work memory released, in MiB. The estimate excludes memory allocator overhead. st.repack.data.set() returns a list with bits.per.genotype, genotype.bytes, dictionary.bytes, roundtrip.verified, and committed. The byte counts describe the proposed or committed allele representation, excluding names, maps, and 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 compiled 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.cross.allocate` *Integer Progeny Allocation to Fixed Crosses*

Description

Allocates a fixed produced-progeny budget across a fixed set of unordered parent pairs, maximizing predicted mean after within-family selection. Retained counts are fixed; produced counts are integer decision variables.

Usage

```
st.cross.allocate(plan, scores, total.progeny, n.selected = 10,
                  min.progeny = n.selected, max.progeny = total.progeny,
                  accuracy = 1, control = list(),
                  selection.method = c("burrows", "order.statistics"),
                  constraints = NULL, genetic.model = NULL)

## S3 method for class 'st.cross.allocation'
print(x, digits = NULL, top = NULL,
                                  bottom = NULL, char.per.line = NULL, ...)
```

Arguments

plan	An st.cross.plan result, or a data frame containing nonempty character columns P1Name and P2Name. Self crosses are excluded. Pair order and orientation follow their first input occurrence. Repeated unordered pairs are pooled as described below.
scores	A data frame with character P1Name, P2Name and finite numeric mu, va columns. Variances must be nonnegative. Every pooled planned pair has exactly one score row; additional unplanned score rows are permitted. Existing es values are not used because selection intensity depends on the new size.
total.progeny	Positive supported whole-number produced budget.
n.selected	Positive retained counts, supplied as a numeric scalar or one value per input plan row.
min.progeny, max.progeny	Positive integer produced-count bounds, each a numeric scalar or one value per input row. Every row satisfies n.selected <= min.progeny <= max.progeny.
accuracy	Finite prediction accuracy in $[0, 1]$, a numeric scalar or one value per input row. Repeated unordered pairs must have identical accuracies.
selection.method	Selection-response calculation. The default "burrows" retains the finite-family approximation used by gs.cross.eval.es. The opt-in "order.statistics" method numerically evaluates the expected mean of the retained top normal order statistics for the actual produced and retained counts.
constraints	Optional practical restrictions described in st.cross.constraints. Explicit restrictions intersect inherited ones. Produced-count restrictions can couple families and require constrained search; the separable allocation optimum then does not certify the constrained problem. Directions, when needed, are identified explicitly.
control	Named list with unique supported components: max.cells Positive integer limit on the estimated DP utility, traceback and numeric-row cells; default 10000000. The order-statistic method estimates its quadrature, cache and allocator workspaces as numeric-equivalent cells. This is a search workspace estimate, not a total byte-memory limit; mandatory output evaluation can still require work after a search limit is reached. max.transitions Positive integer limit on a conservative DP transition-count estimate; default 100000000. For the order-statistic greedy path, the cap counts allocated extra individuals. tolerance Nonnegative response-gap tolerance; default 1e-10. A completed search is qualified optimal when its enclosed response gap is at most this value times max(1, abs(response)). algorithm For selection.method="order.statistics", one of "auto", "greedy" or "dp". The automatic path uses discrete marginal gains; the dynamic program uses the same order-statistic objective. This component is not a Burrows control.
genetic.model	Optional additive DH genetic model, as described in st.cross.evaluate. The model supplies approximate selection-aware founder contributions and optional

final actual-genotype scenarios. An explicit `future` group also requests the final next-cycle assessment described in `st.cross.future`, without changing this allocator's current-generation objective. A source core and cap are inherited with `NULL`; scenario execution always requires a current request. An active `selected.founder.theta` requires an original supplied or inherited `K` and couples the actual produced and retained counts.

`x` An `st.cross.allocation` result.

`digits`, `top`, `bottom`, `char.per.line`
Print controls. `NULL` uses the corresponding `st.options()` setting. Top and bottom counts may be `Inf`.

`...` Reserved for S3 printing.

Details

For pooled family e, r_e is the fixed retained count, g_e the produced count, a_e the accuracy, and $R = \sum_e r_e$. The predicted selected mean is

$$m_e(g_e) = \mu_e + a_e \sqrt{v_e}\, i_B(r_e, g_e),$$

and the objective is $\sum_e (r_e/R) m_e(g_e)$ subject to integer family bounds and a produced budget $\sum_e g_e = B$. The default uses the same finite-family Burrows selection-intensity calculation as `gs.cross.eval.es(N=r_e, G=g_e)`. It uses zero intensity when all produced progeny are retained. Accuracy scales the selection response rather than the family mean.

For `selection.method="order.statistics"`, the intensity is

$$i_O(r, g) = \frac{1}{r} \sum_{j=g-r+1}^{g} E[Z_{(j:g)}],$$

where $Z_{(j:g)}$ is a standard-normal order statistic. This replaces i_B in the selected-mean formula. The finite-normal expectation follows Okoye, Frisch and Herzog (2026), equations (4)–(8). Retained-count weights also cover unequal fixed retained counts; they reduce to the paper's unweighted family average when retained counts are equal. The expectation is exact under the normal scoring model; its numerical evaluation and the fit of that model to biological progeny are separate qualifications. For phenotypic selection, `accuracy` is the square root of heritability under the within-family regression model.

The order-statistic greedy path compares actual one-individual response increments, rather than derivatives of a continuous family-size extension. These increments decrease with produced count for a fixed retained count. Consequently, choosing the largest available weighted increment solves the separable integer problem in exact arithmetic. Bounds truncate the increment sequences. This argument does not extend to joint pair choice, changing retained counts or constraints that couple produced family sizes. The alternative dynamic program evaluates the same requested score.

Repeated reciprocal or identically oriented pair rows describe a single biological selection family. Their retained counts and produced lower and upper bounds are added before evaluating selection intensity. Selection therefore retains the pooled total from the pooled family. It does not perform independent selection in the original repeated rows.

For the default Burrows method, a two-row integer budget dynamic program enumerates all feasible family sizes; integer traceback recovers the allocation. The constant weighted family-mean component is excluded from DP comparisons. No concavity or continuous-size assumption is required.

The algorithm has time complexity $O(FBW)$ and traceback space $O(FB)$, where F is the number of pooled families, B the budget above summed minima, and W the largest feasible size range. These are fixed-pair allocation bounds, not claims of a joint optimum over parent pairs and progeny sizes.

When `plan` is an equal-family `st.cross.plan` result without explicit contribution-basis provenance, nominal retained parental contributions must satisfy its inherited contribution bounds, fixed-contribution requirements and relatedness limit. A retained family contributes half its nominal gene share to each parent. The inherited `K` is used to report retained and produced group relatedness separately. Produced contributions are descriptive and are not constrained by retained-share limits. A family-slot diversity constraint inherited from the original fixed-pair plan is unchanged by altering produced sizes. A plain data frame has no inherited `K` or planning constraints, so its relatedness values are `NA`.

A source plan from the optional `progeny` model explicitly records `family.slots` as the basis of its top-level parental and relatedness restrictions. Those inherited restrictions retain that basis, even when this allocation supplies unequal retained counts. Reported retained and produced contributions and relatedness still use the actual corresponding counts. Explicit practical restrictions retain their own declared bases. The allocator's `total.progeny`, retained counts and produced bounds define a new fixed-pair domain; they replace the source plan's production design, including source equal-size bounds. Its scoring method follows this call's `selection.method`. Original parent, allowed-pair, reproductive role and practical restrictions remain in force.

Neither nominal contribution vector estimates realized genomic ancestry after within-family selection. Crossing populations with the returned variable sizes uses the existing low-level crossing commands; the equal-family execution adapter is unchanged.

A completed DP also propagates an upward addition enclosure over the computed binary64 response table. The reconstructed response receives a downward addition enclosure. Their difference is the reported response gap. This qualification concerns arithmetic on evaluated utilities and does not bound error of normal-distribution or finite-family approximations. Status `optimal` requires completion and the requested gap tolerance; otherwise a completed search has status `numerically.unresolved`. If either resource estimate exceeds its cap, a deterministic feasible allocation fills family capacity in input order and is returned with status `resource.limit`, infinite objective bound and infinite gap. Such a result is an incumbent without an optimality claim.

The order-statistic method additionally reports estimated numerical integration uncertainty and identifies its scoring method and allocator. Status `numerically.optimal` means that the estimated numerical response gap meets the requested tolerance. Quadrature error estimates are not rigorous interval enclosures. A larger unresolved estimate is reported as `numerically.unresolved`; a resource-limited result is reported as `resource.limit`. These qualifications do not certify the finite-normal model for arbitrary biological progeny distributions. A limited greedy search retains the allocated increments and fills the remaining feasible budget in input order. A workspace-limited search can return the initial feasible allocation before greedy increments. The optional genetic model and its inherited selected-founder cap use actual current counts, alongside the original nominal constraints. A cap can couple family sizes even when no other produced/retained practical restriction is active. Its proxy is an explicitly qualified approximation; optional final actual-genotype scenarios describe a separate model-based assessment. `st.cross.evaluate` gives formulas, snapshot inheritance and metric/uncertainty definitions.

Value

An object of class `st.cross.allocation`, containing:

`crosses` One row per pooled family: identifiers, produced and retained counts, `mu`, `va`, `accuracy`, `intensity`, `selected.mean` and pooled produced bounds.

`contributions.retained`, `contributions.produced` Named nominal parental gene shares for the retained and produced samples.

`relatedness.retained`, `relatedness.produced` The corresponding $c'Kc$ quantities, or `NA` without inherited `K`.

`status` Search and numerical qualification.

`objective` Predicted retained mean `value`, weighted `response`, fixed `unselected.mean`, objective upper `bound`, response `gap`, and `bound.type`.

`constraints` Budget, retained total, pooled family count, nominal contribution sample, inherited constraints and re-evaluated `diversity.loss` when an inherited family-slot constraint exists.

`diagnostics` Resource estimates, actual transitions, completion state, pooling count and numerical/ancestry qualifications.

`inputs` Original plan and scores, plus `input.family` mapping original input rows to pooled output families.

`control` Resolved control values.

References

Okoye UJ, Frisch M, Herzog E (2026). Optimization of family sizes in sets of crosses with a greedy allocation strategy based on automatic differentiation. Frontiers in Plant Science 17:1727383. doi:10.3389/fpls.2026.1727383.

See Also

`st.cross.plan`, `gs.cross.eval.es`, `st.cross.progeny`, `st.options`

---

st.cross.constraints *Practical Restrictions for Cross Planning and Progeny Allocation*

Description

The `constraints` argument of `st.cross.plan`, `st.cross.allocate` and `st.cross.evaluate` describes parent counts, reproductive roles, capacities and nominal population restrictions. Every request names its population basis and intersects existing restrictions.

Details

The optional named list has five supported components:

`parents` A table of unique character `parent` IDs, optional character `pool`, and optional logical `seed` and `pollen` eligibility. An omitted eligibility column means that role is allowed. A supplied table covers the candidate universe; reference-only matrix names do not create additional candidates. One pool per parent is supported. Pool labels alone do not declare reproductive direction.

`limits` A table with `parent`, `pool`, `role`, `basis`, `measure`, `lower` and `upper`. A missing parent selects a group; a missing pool selects all pools. Roles are `"all"`, `"seed"` and `"pollen"`. Measures are `"parents"` for distinct used parents, `"uses"` for integer incidences, and `"share"` for normalized contribution. Parent counts apply to groups. Equal lower/upper bounds specify exact limits. Integer bounds use supported whole numbers; shares lie in $[0, 1]$. Repeated identical selectors are errors.

`equal.use` A table of `pool`, `role` and `basis`. All actually used parents in each selected subset have equal integer incidence. It does not require every eligible parent to be used; a distinct-parent limit specifies that requirement separately.

`relatedness` A table of `basis` and `max`, imposing $c'Kc <= \theta$ on overall nominal shares with the original named `K`. It does not replace the existing `theta` argument.

`diversity` A list with `G`, `max` and `sample`. Samples are `"family.slots"`, `"produced.founder.slots"` and `"retained.founder.slots"`. Literal integer founder multiplicities and the complete fixed covariance reference are used by `propsd`. Its estimated proportional loss is strictly below `max`. This request intersects the existing family-slot diversity restriction.

The supported incidence bases are `"family.slots"`, `"produced"`, `"retained"` and `"operations"`. For produced, retained or identified operation weights w_a, with $T = \sum_a w_a$, seed and pollen incidences are

$$S_i = \sum_{a:\mathrm{seed}(a)=i} w_a, \qquad L_i = \sum_{a:\mathrm{pollen}(a)=i} w_a.$$

Overall gene share is $(S_i + L_i)/(2T)$; seed and pollen shares are S_i/T and L_i/T. A pool sums these quantities with the same overall or role denominator; it is not normalized within its own pool. An overall distinct-parent count counts a parent once even when it occurs in both roles. Role-specific counts count its use in each selected role.

Family slots count each active unordered biological family once, regardless of duplicated or reciprocal rows. Its overall denominator is twice the number of active families. Role-specific family slots require an unambiguous direction for every active family. Multiple reproductive records can contribute to one pooled biological response, but cannot create additional families or selection response by duplication.

Generated role-aware plans and allocations carry a `directions` table with `family`, `seed` and `pollen`, without duplicated progeny counts. Its actual unordered parent key joins to the current `crosses` counts. An old role-free object's display order does not identify reproductive direction. A raw data frame with an explicit new role contract declares `P1Name` as seed and `P2Name` as pollen. Its original reciprocal rows retain their separate actual counts for reproductive accounting; response pools by biological family.

Operation multiplicities are identified only by explicit `n.operations`; they are not inferred from progeny counts. Each is a nonnegative exact whole number at most $2^{53} - 1$, and their total obeys the

same limit. A positive produced count requires positive identified operation count when that basis is requested. Zero production with recorded positive operations describes an unsuccessful operation. The current planner and fixed-family allocator do not optimize operation multiplicities or reciprocal production splits.

Produced and retained counts keep the selection kernel's per-family integer domain. Group incidence can exceed $2^{53} - 1$; feasibility and exact character reports retain its integer value. Numeric report fields are unavailable when the value cannot be represented in the supported exact numeric domain. Share checks use exact incidence ratios and the represented numeric bounds. Integer equal-use restrictions describe exact equal use. The literal multiplicity total for a PropSD sample is separately limited to $2^{53} - 1$; a larger sample is not silently rescaled.

No-selection and zero-total situations retain their declared meanings. Shares or relatedness with a zero denominator are unavailable. The equal-family planner has no retained or operation counts to infer, so restrictions requiring those counts are rejected. Fixed-family allocation with produced-share, capacity, relatedness or produced-diversity coupling uses a constrained search. The separable greedy/dynamic-programming optimality claim does not transfer to that problem. Search limits and unresolved numerical boundaries remain explicit.

Nominal founder-slot PropSD and $c'Kc$ are not actual selected-DH genomic diversity.

Value

This topic describes an argument rather than a separate function. Opted-in results retain effective practical restrictions under `constraints$practical`; the current explicit request remains under `inputs$practical`. Recomputed checks are in `diagnostics$practical`. Evaluation reports independently recomputed feasibility. Reports distinguish passed, failed, numerically unresolved and unavailable restrictions; a saved source status does not establish feasibility or optimality.

References

Gorjanc G, Hickey JM (2018). AlphaMate: a program for optimizing selection, maintenance of diversity and mate allocation in breeding programs. Bioinformatics 34:3408–3411. doi:10.1093/bioinformatics/bty375. The associated manual's named mating/crossing-constraint sections describe parent counts, role-specific counts, equal contributions and contribution bounds: https://github.com/AlphaGenes/AlphaMate/blob/main/doc/AlphaMate.txt.

See Also

`st.cross.plan`, `st.cross.allocate`, `st.cross.evaluate`, `propsd`

`st.cross.evaluate` *Evaluation of a Produced and Selected Progeny Population*

Description

Evaluates the actual integer family sizes in a cross plan without changing them. Population summaries distinguish produced progeny from retained progeny and nominal parental contributions on three separate bases.

An optional additive DH genetic model supplies selection-aware founder proxies and actual-genotype assessments. Its `future` group describes one additional crossing and selection cycle.

Usage

```
st.cross.evaluate(plan, scores = NULL, n.selected = NULL,
                  accuracy = NULL, K = NULL, selection.method = NULL,
                  constraints = NULL, genetic.model = NULL)

## S3 method for class 'st.cross.population'
print(x, digits = NULL, top = NULL,
                                  bottom = NULL, char.per.line = NULL, ...)
```

Arguments

plan	An st.cross.plan, st.cross.allocation, or data frame with character P1Name, P2Name and integer-valued n.progeny columns. Unordered repeated families are pooled before selection response is evaluated. Self crosses are excluded.
scores	Optional keyed data frame with P1Name, P2Name, finite numeric mu and nonnegative finite va. Otherwise the plan's stored score columns are used. Conflicting scores for repeated families are errors.
n.selected	Nonnegative retained counts, supplied explicitly as a scalar or one value per input row, or taken from the stored column. Each count is no larger than its produced count.
accuracy	Prediction accuracy in $[0, 1]$, supplied as a scalar or one value per input row. A stored column is used when available, followed by a source cross plan's stored scalar accuracy; otherwise the value is one. Repeated families have identical accuracy.
K	Optional full named symmetric positive-semidefinite parental relationship reference, or the inherited reference. Unused named parents have zero contributions. Its original scale is preserved. An explicit reference conflicting with inherited constraints is an error.
selection.method	"burrows" or "order.statistics". NULL uses the source allocation's method when stored and the source plan's explicit progeny$selection.method for that model, with the Burrows approximation otherwise. The latter choice preserves the selection calculation of gs.cross.eval.es; normal order statistics use the finite-family numerical evaluator of st.cross.allocate.
constraints	Optional practical restrictions described in st.cross.constraints, intersecting supported inherited restrictions. Evaluation checks actual counts and preserves the source optimizer's status separately.
genetic.model	Optional additive DH model. A fresh named list has data.set, population="DH", t=0, map.function="Haldane", index.model="trait.independent.noise", marker.weights=NULL, selected.founder.theta=NULL, scenarios=NULL, future=NULL and control=list(max.cells=1e7,max.work=1e8). Only data.set is required for a fresh model; the other values are defaults. Named nonnegative marker weights cover every model marker and are normalized once. The model requires complete homozygous biallelic candidate parents, complete reference genotypes, fitted additive effects, dosage coding 0/1/2 and a complete map in cM. The optional finite nonnegative selected.founder.theta caps the approximate selected-founder quadratic on the original supplied or inherited K.

`scenarios=list(replicates=S,seed=0L)` requests final actual-genotype Monte Carlo evaluation, with positive whole `S` and a nonnegative supported whole-number seed. Genetic work and memory controls are cumulative within the call. An explicit `future` group requests actual next-cycle DH assessment described in `st.cross.future`. `NULL` inherits an actual source object's frozen core and cap, without executing its old scenario request. A list without `data.set` can request new scenarios, future assessment, controls or a tighter inherited cap. Conflicting cores and unknown or repeated fields are errors. The details below define the model and available summaries.

`x` An `st.cross.population` result.

`digits`, `top`, `bottom`, `char.per.line`
Print controls. `NULL` uses the corresponding `st.options()` setting.

`...` Reserved for S3 printing.

Details

For family e, produced and retained counts are n_e and r_e, with totals P and R. The produced population has

$$M_0 = \sum_e (n_e/P)\mu_e, \qquad V_0 = \sum_e (n_e/P)v_e + \sum_e (n_e/P)(\mu_e - M_0)^2.$$

The first variance term is within families; the second is between families. They describe marginal individual variance, not the expected empirical sample variance.

The expected selected-family mean is

$$U_e = \mu_e + \rho_e \sqrt{v_e}\, a(n_e, r_e),$$

where a is the selected finite-family intensity. The selected population mean is $J_1 = \sum_e (r_e/R)U_e$; its response is retained separately from the retained-weight baseline $\sum_e (r_e/R)\mu_e$. The between-family term $\sum_e (r_e/R)(U_e - J_1)^2$ is reported. Without an optional genetic model, the selected-population summary contains the expected mean and its between-family component. It does not provide selected within-family or total variance, realized genomic ancestry or selected genomic diversity. Additional-cycle assessment requires the explicit `future` group described below.

Nominal contributions count one half from each parent, weighted either by active unique families (`family.slots`), produced counts (`produced`), or retained counts (`retained`). They are not selection-adjusted genomic ancestry. Relatedness on each available basis is $c'Kc$. Produced counts do not identify repeated crossing operations.

Zero-produced families are inactive. A produced family retaining no progeny has unavailable intensity/selected mean and zero retained weight. Summaries whose denominators are zero remain unavailable with a reason. Counts and pooled counts lie between zero and the supported C integer maximum; totals are checked for exact representation.

Inherited supported budgets, bounds and diversity restrictions are checked again against actual counts. A structurally valid violation is returned as infeasible, with residuals; an optimizer's saved status is not proof of feasibility. Unsupported inherited restrictions remain unqualified. The evaluator neither reallocates progeny nor certifies optimality. Numerical order-statistic error estimates are not rigorous interval bounds.

A direct progeny source plan is checked against its declared active family count, produced budget and count bounds. Fixed retention supplies a common retained count; variable retention supplies a total retained budget and biological-keyed produced/retained family bounds. Zero-retention produced families remain active on the family and produced bases. An allocation descended from that plan instead has its own direct production and retention domain. Original source parental, pairing and practical restrictions remain inherited. Explicit source contribution and relatedness bases are preserved; actual produced and retained summaries remain separate. With genetic.model, a frozen versioned snapshot stores the candidate marker genotypes, fitted effects, map, weights and fixed genomic reference. Subsequent source evaluation does not require the live dataset. Preparation reads the marker data without changing its datasets, caches or simulation state. A conflicting explicit K remains an error; the inherited original reference and its scale remain fixed. A warm-start plan supplies pair/count candidates, without implicitly supplying a new planner's model.

Here x_{pj} is parent p's effect-allele presence and u_j is the stored dosage coefficient. Parental value is $u_0 + 2\sum_j u_j x_{pj}$. For an immediate DH family, $d_j = u_j(x_{1j} - x_{2j})$ and $G = \mu + d'S$, where parental origin S_j is $+1$ or -1. The origin covariance is $Q_{jk} = \exp(-2|m_j - m_k|)$ on the same chromosome, with positions m in Morgans; it is zero between chromosomes. Thus $v = d'Qd$, weighted P1 origin is $C_1 = (1 + w'S)/2$, and $h = \mathrm{Cov}(C_1, G) = w'Qd/2$. Positive linear-time Haldane recurrences prepare these family moments without a dense marker matrix.

For positive retained count, the selected-origin approximation is

$$c_1 = \frac{1}{2} + \rho\, a(n, r) h/\sqrt{v}, \qquad c_2 = 1 - c_1.$$

This adapts Allier et al. (2019), G3 equations (4)–(6), and Frontiers in Genetics equations (5)–(7), using the requested finite-family intensity. It assumes independent noise in the declared selection index; accuracy alone does not identify an arbitrary index's ancestry response. Full retention, zero accuracy and zero trait variance give nominal one-half contributions. Zero-retention families have no conditional selected contribution and have zero aggregate weight. The retained-weight aggregate gives the ucpc.founder.proxy quadratic $c'Kc$, corresponding to the plug-in diversity proxy in Frontiers in Genetics equation (9). It omits finite-panel contribution covariance and locus-specific shifts. It is neither actual allele coancestry nor an input multiplicity for propsd. Raw fractions outside $[0, 1]$ remain visible and make the qualified approximation unavailable; they are never clipped. In-range predictions can also be biased. An unresolved approximation cannot prove infeasibility. Estimated numerical errors do not bound this biological approximation bias. Without a cap these predictions do not change the search.

Optional scenarios generate actual immediate-DH marker-origin paths under Haldane meiosis and select using $I = \rho(G - \mu)/\sqrt{v} + \sqrt{1 - \rho^2}\,\epsilon$, with independent standard-normal noise and neutral ties. Direct genetic deviations preserve ranking and response in the presence of a large common trait intercept. The private common bank preserves candidate prefixes and leaves the caller's R and simulation random streams unchanged. It runs at the final assessment, not at each optimizer count candidate. No scenario score is inherited as a prediction for changed counts.

With fixed reference frequencies p_{0j} and normalized weights, $D = \sum_j w_j p_{0j}(1 - p_{0j})$. Each selected replicate has

$$q = \frac{\sum_j w_j (p_j - p_{0j})^2}{D}, \qquad H = 2\sum_j w_j p_j (1 - p_j).$$

The genomic.q metric uses weighted VanRaden A/2 with all line pairs, including the diagonal; st.get.A itself returns A. The selected population is never used to recenter its reference. gene.diversity

is population allele diversity; individual DH heterozygosity is zero. Nonlinear metrics are calculated within each replicate before averaging. A zero reference denominator makes only the genomic quadratic unavailable. Positive normalizers or marker weights that cannot be represented consistently within the model's numerical precision produce a `numeric.range` error.

Scenario variance summaries distinguish marginal selected-member variance, expected finite-panel variance with denominator R, and expected sample variance with denominator $R-1$. The marginal estimate is the mean finite-panel variance plus the unbiased variance of replicate panel means; it requires multiple completed replicates. Its SE is explicitly unavailable. Other available SE values use independent whole-replicate summaries. One retained individual has finite-panel variance zero and unavailable sample variance; one replicate has unavailable SE. Empty selected populations have reasoned unavailable selected summaries and no candidate draws. Resource limits publish only whole completed replicates and identify a partial bank. These fixed-model simulations describe the immediate DH generation; they do not provide independent field validation.

Value

An object of class `st.cross.population`, with normalized `crosses`, `population` summaries, `contributions`, `relatedness`, `feasibility`, `availability`, `diagnostics` and declared `inputs`. Original input-row mapping and source status remain available. The print method returns its input invisibly.

The `future` group is a named input rather than a separate R function. Activated planning and assessment results contain a `future` group with the horizon, policy and resources, training and evaluation identities, completion status, current and next selected-population summaries, conditional variance/SD, paired comparator differences and resource diagnostics. Their exact availability depends on completed evaluations. Existing current-generation analytic fields retain their meanings. For an explicit assessment, `population$future.mean` contains the same available held-out G2 mean, and `availability$G2` and its reason describe its availability and completion. Uncertainty remains in the dedicated `future` summaries.

Additional DH cycle

The optional `genetic.model$future` group supports these named components:

`policy` Mating policy `"random.disjoint"`.

`n.progeny`, `n.selected` Positive whole-number produced and retained counts per next-cycle family, defaulting to 20 and 2, respectively.

`accuracy` Selection accuracy between zero and one, defaulting to one.

`training` Named list of `replicates` and `seed`, defaulting to 32 and zero.

`evaluation` Named list of `replicates` and `seed`, defaulting to 256 and one.

`comparators` Optional named list of actual plans, allocations, population evaluations or count tables; the default is `NULL`.

This group belongs to the additive inbred DH model described above. Current production and retention counts come from the evaluated plan, allocation or count table; joint planning budgets are described in `st.cross.progeny`. The selected current population contains an even number of at least two logical DH lines. Uniform random disjoint matching uses every selected line exactly once. Distinct lines with identical genotypes or from the same original family may mate. New F1 crosses

between those lines produce the next DH families. A homozygous DH line has no segregation variance by itself.

The future family counts satisfy $n_2 \geq r_2 \geq 1$. Future selection accuracy is between zero and one. The number of next-cycle families is half the current retained total; their produced and retained totals are that number times `n.progeny` and `n.selected`, respectively. Counts and replicate numbers are supported positive whole numbers; seeds are supported nonnegative whole numbers. Unknown, repeated and malformed fields produce errors.

The model uses the current frozen marker effects, actual parental haplotypes, Haldane recombination and independent selection-index noise in both cycles. Conditional next-cross variance is calculated from the actual selected genotypes. Mean conditional variance and mean conditional SD are separate summaries; neither a kinship matrix nor the square root of mean variance replaces those genotypes.

`criterion="future"` requires joint planning, an explicit `progeny` group and a valid requested genetic model. Its objective is the mean genetic value in the next selected population. Search compares only complete evaluations on a fixed training bank. A separate evaluation bank assesses the chosen plan. Training and evaluation streams differ even when their user seeds are equal. A finite-bank optimum is not an optimum of the unknown expectation. Larger searches retain their best-found status and unresolved gap. Existing current-generation feasibility restrictions, including an active selected-founder proxy cap, still apply to actual current counts. Such a proxy cap is not a bound on actual next-generation genomic coancestry.

An explicit `future` group with an ordinary mean or usefulness plan, `st.cross.evaluate`, or `st.cross.allocate` requests a final assessment only. It does not alter that operation's current-generation objective or search path. Odd or insufficient current retention yields an unavailable future assessment without changing current feasibility; future optimization rejects those retention totals before search.

`comparators` is an optional named list of actual plans, allocations, population evaluations or count tables. Names are unique and nonempty; `selected` and `G1.reference` are reserved. Comparators are checked against the same genetic core, current resources and applicable current restrictions. Unavailable or invalid comparator status is individual. It cannot change the chosen plan's feasibility. Future planning also retains its checked current-generation reference. Chosen-minus-comparator differences and SE use whole shared evaluation replicates. Evaluation outcomes do not rank search proposals or select later warm starts.

Normalized model metadata retains only the future mating policy, family counts and accuracy. A new explicit request is required to execute an assessment. Banks, comparator commands, results and certificates from a source object are not inherited. The frozen genetic snapshot and its signature are unchanged by future policy metadata. Planner starts contribute counts and pairs, without supplying an implicit model or future policy.

One cumulative genetic cell/work budget covers preparation, caches, training and assessment, including all frontier points. A finite assessment allowance is reserved before search; unusual costs can still leave it incomplete. Requested and completed whole replicates and the resource status remain explicit. Partial training banks do not compete with complete candidates. Genomic coancestry, gene diversity and variance summaries use the same fixed-reference definitions as the current DH assessment, with nonlinear statistics calculated per replicate.

The forecast is conditional on fitted effects, the declared selection model and one further cycle. Simulation agreement is model consistency, not independent field validation or an indefinite breeding guarantee.

References

Okoye UJ, Frisch M, Herzog E (2026). Optimization of family sizes in sets of crosses with a greedy allocation strategy based on automatic differentiation. Frontiers in Plant Science 17:1727383. doi:10.3389/fpls.2026.1727383.

Allier A, Moreau L, Charcosset A, Teyssedre S, Lehermeier C (2019). Usefulness Criterion and Post-selection Parental Contributions in Multi-parental Crosses: Application to Polygenic Trait Introgression. G3: Genes, Genomes, Genetics 9:1469–1479. doi:10.1534/g3.119.400129.

Allier A, Lehermeier C, Charcosset A, Moreau L, Teyssedre S (2019). Improving Short- and Long-Term Genetic Gain by Accounting for Within-Family Variance in Optimal Cross-Selection. Frontiers in Genetics 10:1006. doi:10.3389/fgene.2019.01006.

Niehoff TAM, ten Napel J, Bijma P, Pook T, Wientjes YCJ, Hegedűs B, Calus MPL (2024). Improving selection decisions with mating information by accounting for Mendelian sampling variances looking two generations ahead. Genetics Selection Evolution 56:41. doi:10.1186/s1271102400899-2. Equations (2), (8), and (9) motivate the conditional-variance distinction; the DH cycle here uses actual selected genotypes.

Hamazaki K, Iwata H (2024). AI-assisted selection of mating pairs through simulation-based optimized progeny allocation strategies in plant breeding. Frontiers in Plant Science 15:1361894. doi:10.3389/fpls.2024.1361894. Sections 1 and 2.1 concern optimization of future progeny populations.

Niehoff TAM, ten Napel J, Calus MPL (2025). Practical Considerations When Using Mendelian Sampling Variances for Selection Decisions in Genomic Selection Programs. Journal of Animal Breeding and Genetics 142:419–437. doi:10.1111/jbg.12913. Sections 4.1 and 4.4 discuss accuracy and model-explainable variance under estimated marker effects.

See Also

`st.cross.allocate`, `st.cross.plan`, `st.cross.progeny`, `propsd`, `st.cross.frontier`, `st.cross.future`

`st.cross.plan` *Cross Planning, Progeny Budgets and Gain-Diversity Frontiers*

Description

Optimum cross selection chooses distinct non-selfed parent pairs with, by default, equal family sizes, contribution bounds and a group-relatedness limit. Joint planning maximizes additive mean; an optional cross-score table also supports usefulness-based joint planning. Fixed-contribution planning minimizes average relatedness within crosses. The optional `progeny` group jointly chooses pairs and produced counts, with either fixed family retention or an explicit total retained budget.

Cross-planning frontiers report assessed plans, feasibility and numerical status at a declared sequence of diversity limits. They support both current-generation and additional-cycle population objectives.

Usage

```
st.cross.plan(merit, K, n.crosses, theta, n.progeny = 100,
              lower = 0, upper = 0.5, pairs = NULL,
              contributions = NULL, mode = "joint", control = list(),
              scores = NULL, criterion = "mean", accuracy = 1,
              diversity = NULL, constraints = NULL, progeny = NULL,
              genetic.model = NULL)

## S3 method for class 'st.cross.plan'
print(x, digits = NULL, top = NULL, bottom = NULL,
                           char.per.line = NULL, ...)

st.cross.frontier(merit, K, n.crosses, theta, n.progeny = 100,
                  lower = 0, upper = 0.5, pairs = NULL,
                  contributions = NULL, mode = "joint", control = list(),
                  scores = NULL, criterion = "mean", accuracy = 1,
                  diversity = NULL, constraints = NULL, progeny = NULL,
                  genetic.model = NULL, limits, metric = "theta",
                  warmstart = FALSE)

## S3 method for class 'st.cross.frontier'
print(x, digits = NULL, top = NULL,
                                bottom = NULL, char.per.line = NULL, ...)
```

Arguments

`merit`	Finite numeric vector of additive parental merits, with at least two parents. Its order agrees with both dimensions of `K`. Names, when supplied, are unique, complete parental IDs.
`K`	Finite symmetric positive-semidefinite relationship matrix in parental order. Singular matrices are supported. For an additive relationship matrix `A`, the kinship convention is `K = A/2`. Row and column names are either both absent or both supplied and equal. Named `merit` entries agree with these names in the same order. Parental IDs follow `merit` names, otherwise `K` names, otherwise the character labels `"1"`, `"2"`, and so on. With practical `constraints`, a full named reference may include additional unused parents; named candidate merits identify the planning subset. The full reference remains in `inputs$K`, and unused reference shares are zero. The default role-free matrix/order contract remains unchanged.
`n.crosses`	Positive supported whole number of distinct families. Each unordered parent pair occurs at most once, and self crosses are excluded.
`theta`	Finite upper limit on group relatedness, $c'Kc$, on the original reference scale of `K`.
`n.progeny`	Positive supported whole number of progeny per family. All families have this same size on the equal-family path.
`lower, upper`	Lower and upper bounds on parental gene contributions. Each is an unnamed numeric scalar or a numeric vector matching `merit`. Full-length names, when

supplied, match the parental order. Bounds satisfy `0 <= lower <= upper <= 1`; a non-selfed plan can use no parent in more than half of its total parental gene contribution.

`pairs` Allowed unordered pairs, supplied as a two-column character matrix or a data frame with exactly two character columns containing parental IDs. Column order specifies the two parents; column names are not used. Unknown IDs, self pairs and repeated unordered pairs produce errors. `NULL` allows every non-selfed pair. Reciprocal rows describe the same allowed pair under the default role-free model. With explicit reproductive roles in `constraints`, column one identifies the seed parent and column two the pollen parent; reciprocal directions are alternatives for one biological family.

`contributions` Required only for `mode = "fixed"`. A finite numeric vector in parental order specifying nonnegative gene contributions that sum to one. Full-length names, when present, match the parental IDs. Each contribution must be realizable as an integer number of parental uses divided by `2 * n.crosses`, and the resulting uses must satisfy the contribution bounds and allowed-pair restrictions.

`mode` Either `"joint"`, which jointly chooses parental usage and pairing under the selected criterion, or `"fixed"`, which preserves supplied contributions and minimizes average within-cross relatedness. Fixed mode requires `scores = NULL` and `criterion = "mean"`.

`control` Named list of search controls. Unknown and repeated control names produce errors. The supported components are:

`method` `"auto"` by default, or `"exact"` or `"heuristic"`. Automatic selection uses the exact search when its estimated work fits the node budget, and the heuristic otherwise.

`max.nodes` Positive whole-number exact-search node budget; default `1000000`.

`maxit` Positive whole-number proposal budget per heuristic start, including budgeted degree-realization attempts and local changes; default `20000`.

`starts` Positive whole number of heuristic starts; default `16`.

`seed` Nonnegative supported whole-number seed for the search's local random-number stream; default `1`.

`tolerance` Finite number strictly between zero and one; default `1e-8`. Controls numerical comparisons. The relatedness allowance uses the active quadratic terms and requested limit, as described below. Reported objective and relatedness values use the original input units.

`start` Optional earlier `st.cross.plan` result or its `crosses` data frame, containing a feasible incumbent with the requested family count and progeny count, whose pairs satisfy the current allowed-pair restrictions and constraints. Its actual family rows are validated before use. This starts a new search from an incumbent; it is not a continuation at an earlier saved random-number state. The `progeny` path also accepts an allocation as a count candidate and independently rescores it under current inputs.

`scores` Optional data frame with character columns `P1Name` and `P2Name`, and numeric columns `mu`, `va`, and `es`. These contain cross means, nonnegative segregation variances, and expected superior-progeny scores, as returned by `gs.cross.info` after the corresponding evaluations. All score values are finite. Each allowed

pair has exactly one score row. Extra rows are permitted but all rows contain known, different parents and unique unordered pairs. Row order and parental order within a pair are immaterial. Additional columns are permitted. The table is required for usefulness planning. With `progeny`, only `mu` and `va` are required and used; the selected score is recomputed for actual counts.

`criterion` Either `"mean"`, `"usefulness"`, or `"future"`. Without `scores`, mean planning uses the mean of parental merits. With `scores`, mean and usefulness planning use the supplied cross means, and usefulness includes the supplied expected within-family selection gain on the equal-family path. With `progeny`, usefulness uses the requested count-dependent selection calculation. Future planning requires an explicit `progeny` group and genetic model, and targets the actual next selected DH population described in `st.cross.future`.

`accuracy` Finite within-family selection accuracy between zero and one. The usefulness score is `mu + accuracy * (es - mu)`. The default of one retains the supplied `es`; zero gives the supplied `mu`. A nondefault value requires `scores`. With `progeny`, accuracy multiplies the recomputed finite-family selection differential.

`diversity` Optional list with exactly `G`, `max`, and `sample = "family.slots"`. The sample declaration is mandatory. `G` is the complete named fixed reference covariance matrix accepted by `propsd`; it contains every planning parent and may contain additional reference parents. `max` is a finite strict upper limit on the computed proportional loss. `NULL` retains the unconstrained interface and output structure.

`genetic.model` Optional additive DH genetic model requiring an explicit `progeny` group, as described in `st.cross.evaluate`. The model supplies approximate selection-aware founder contributions and optional final actual-genotype scenarios. Its optional `future` group supports actual next-cycle assessment and planning as described in `st.cross.future`. A supplied frozen snapshot supports use without a live marker data set. Planner starts supply counts and pairs, without an implicit genetic model or future policy. Scenario execution always requires a current request. An active `selected.founder.theta` requires the original supplied `K` and couples the actual produced and retained counts.

`x` A cross-planning result or cross-planning frontier result.

`constraints` Optional practical parent, pool, role and resource restrictions described in `st.cross.constraints`. They intersect the existing contribution, pair, relatedness and diversity restrictions. Retained-count and operation-count restrictions require identified counts; the equal-family path does not infer them.

`progeny` Optional joint family-size specification described in `st.cross.progeny`. `NULL` retains the equal-family planner. The group supplies a produced budget, fixed common retained count or variable-retention total, supported family bounds, selection method and allocation controls. It requires `mode="joint"` and keyed `mu`/`va` scores. An explicit top-level `n.progeny` conflicts with this group.

`digits`, `top`, `bottom`, `char.per.line`
Optional printing controls. Unspecified values follow the corresponding `st.options` settings.

`...` Additional printing arguments.

`limits` Nonempty numeric vector of finite declared limits, in the requested point order. Selected-founder proxy limits are nonnegative.

`metric` Either `"theta"` for original family-slot group relatedness or `"selected.founder.theta"` for the selection-aware founder proxy. Each point replaces the parameter named by `metric`; independent restrictions continue to intersect it. The latter metric requires a valid genetic model and original relationship reference. Neither is an actual future-genomic bound.

`warmstart` Logical scalar. The default `FALSE` uses each point's standalone starting rules and seed. With `TRUE`, a checked previous plan can supply actual pairs and counts as a start; acceptance, rejection and fallback remain explicit. Held-out results do not choose warm starts.

Details

The following equal-family score formulas describe `progeny=NULL`. The optional joint produced-budget model and its count-dependent formulas are described in `st.cross.progeny`. Relatedness and top-level parental bounds retain their explicit family-slot basis on that path.

For m equal-sized families, a parent used in d_i families has nominal gene contribution $c_i = d_i/(2m)$. Each selected pair contributes one use to each of its two parents. The contributions sum to one. Equal positive family sizes make family shares and produced-progeny shares equal; changing the common family size does not change the selected-pair objective.

In joint mode the objective is

$$\frac{1}{m}\sum_{(i,j)\in S}\frac{g_i+g_j}{2} = g'c,$$

where S contains the selected pairs and g is `merit`. The planner chooses both usage and pairing under $c'Kc \le \theta$ and the contribution bounds. Integer usage bounds follow from the supplied fractional bounds, without allowing extra uses beyond those bounds.

When `scores` is supplied, joint mean planning instead maximizes the average supplied `mu` over selected pairs. Usefulness planning maximizes the average

$$U_{ij} = \mu_{ij} + r(es_{ij} - \mu_{ij}),$$

where r is `accuracy`. Both criteria retain the same constraints and report both mean and usefulness for their selected pairs. The supplied cross means need not equal means reconstructed from exported parental merits. `parent.gain` separately reports $g'c$ in this case.

The upstream functions `gs.cross.eval.mu`, `gs.cross.eval.va`, `gs.cross.eval.es`, and `gs.cross.info` supply one common score table for comparisons. `gs.cross.eval.es(N = s, G = n)` incorporates the existing finite-family approximation of Burrows (1972); its `N` and `G` are retained and produced individuals per family. The planner does not recalculate this score when `n.progeny` changes. The family size, retained count, progeny type, marker-effect model and selection convention represented by the table therefore form part of the declared analysis. In particular, usefulness is a fixed cross score in this interface, rather than an allocation of varying family sizes.

The table requires `es >= mu`, apart from arithmetic-precision noise; `control$tolerance` does not relax score validation. Supplied score values are retained without rounding or replacement. An accuracy of zero, or an upstream score with `es = mu`, reduces usefulness planning to mean planning on the same table. Fixed cross scores leave the objective linear in pair selection; they do not change the interpretation of the group-relatedness constraint.

In fixed mode the objective is

$$\frac{1}{m}\sum_{(i,j)\in S} K_{ij}.$$

Both $g'c$ and $c'Kc$ are fixed by the supplied contributions. Re-pairing those contributions can reduce average within-cross relatedness, but cannot change the overall additive mean. Arbitrary continuous contributions from `st.ocs` need not correspond to an integer plan. Machine-arithmetic noise near an integer use count is tolerated; other nonintegral requests are reported as infeasible rather than silently rounded. Increasing `control$tolerance` does not change this integer-realizability requirement. Integer uses also need a realization with distinct allowed non-selfed pairs.

The matrix and its allele-frequency reference remain fixed throughout the search. A centered genomic matrix describes a relatedness index relative to that reference. Dividing an additive relationship matrix by two does not itself provide an absolute probability of identity by descent. The `relatedness` limit and the average `within.cross.relatedness` are separate quantities.

Relatedness is recomputed from the original `K` and integer parental uses. Its numerical feasibility allowance is

$$\text{tolerance}\,\max\left(|\theta|,\sum_{i,j}|K_{ij}|c_ic_j\right).$$

Matrix entries for unused parents do not create an extra allowance. The entire computed roundoff interval must satisfy the bound including this allowance; an interval that crosses its boundary remains unresolved. An exact zero limit additionally requires an arithmetic-precision check of the original matrix rows applied to c. This zero check is independent of `control$tolerance`. Numerically unresolved boundary points do not establish infeasibility or optimality; a result may retain `"search.limit"` or `"best.found"` for this reason.

With `diversity` enabled, each chosen family contributes one occurrence of each parent to the selected multiset used by `propsd`. Thus parent i has multiplicity d_i and there are $2m$ occurrences. This family-slot convention is explicit and does not represent produced progeny counts, retained descendant counts or continuous gene shares. The full reference matrix is retained, including unused reference parents. The constraint is the strict computed comparison `loss < max`; the search comparison tolerance does not relax it. This constrains a Gamma approximation, not a formal interval enclosure of biological diversity loss. The statistic can be negative and is not clipped. It remains separate from both relatedness quantities and from the mean or usefulness objective.

The exact search checks diversity at each complete candidate; its unconstrained objective bounds remain valid but may be loose. The heuristic constructions can initially be infeasible. In joint mode, repair uses both relatedness and diversity violations, and a move from a feasible state must preserve both computed constraints. Only independently rechecked feasible incumbents are retained. In fixed mode, multiplicities and therefore PropSD are fixed, so a violated diversity limit establishes infeasibility before rewiring. Failure to find a feasible joint plan heuristically remains a search limit.

The exact method uses a bounded exhaustive search with pruning. An optimal status requires a completed search with a feasible optimum certified under the stated floating-point comparisons. A search stopped by the node budget does not receive that status. The heuristic uses multiple constructions and local changes while retaining only independently rechecked feasible plans. In fixed mode, local rewiring preserves parental use counts. A successful greedy construction or feasible warm start proceeds directly to local search. If construction fails before a feasible incumbent is available, randomized degree realization can use the remaining proposal budget for that start. Each

attempt counts against `control$maxit`; these attempts provide no optimality certificate. Failure to find a feasible plan within a search budget is distinguished from proven infeasibility.

Mean and usefulness describe the declared additive model. Parental contributions remain nominal shares before within-family selection. Realized ancestry after selecting descendants is not determined by these contributions alone. An equal-family plan can be executed with `st.cross.plan.execute` when the selected founders satisfy that function's simulation requirements. Varying actual family counts are available to the individual simulation commands.

Inputs, the caller's R and simulation random-number states, and thread settings are preserved by the planner. The local search seed and final state are returned with the result. The numeric input precision is retained. The optional genetic model and its inherited selected-founder cap use actual current counts, alongside the original nominal constraints. A cap can couple family sizes even when no other produced/retained practical restriction is active. Its proxy is an explicitly qualified approximation; optional final actual-genotype scenarios describe a separate model-based assessment. `st.cross.evaluate` gives formulas, snapshot inheritance and metric/uncertainty definitions.

Value

A list of class `st.cross.plan`. Its printed representation summarizes the objective, feasibility, families and parental usage under the current printing options. The principal components are:

`crosses` Data frame with columns `family`, `P1Name`, `P2Name`, `n.progeny`, `mu`, and `kinship`. Without a score table, `mu` is the mean of the two supplied parental merits. With a score table it is the supplied cross mean, and the table also contains `va`, `es`, and `usefulness`. `kinship` is the pair's entry in the supplied `K`. The table has zero rows when no feasible plan is returned. With `progeny`, the table contains actual `n.selected`, selection `accuracy`, `intensity`, `selected.mean` and `response`; stored `es` does not supply its count-dependent utility.

`contributions` Named numeric vector of achieved parental gene shares, or `NULL` when no feasible plan is returned.

`parents` Data frame in parental order, containing `parent`, `uses`, `contribution`, `lower`, `upper`, and `merit`, including unused parents.

`gain` Achieved mean merit: $g'c$ without a score table, or the average supplied cross mean with a score table.

`usefulness`, `parent.gain`, `criterion`, `accuracy`
Additional components when a score table is supplied: average achieved usefulness, parental merit $g'c$, the requested criterion and within-family accuracy.

`diversity` Present only when enabled. Contains the requested `max` and `sample`, achieved `loss`, a `feasible` indicator, and `evaluation`, the complete `propsd` result. When no feasible plan is returned, loss is `NA`, feasibility is false and evaluation is `NULL`. The request is also retained in `inputs$diversity`, with the threshold and sample declaration in `constraints$diversity`.

`relatedness` Achieved group relatedness, $c'Kc$.

`within.cross.relatedness`
Mean K_{ij} over the selected pairs.

`mode` The requested planning mode.

`status` "optimal" for a completed exact search with a certified feasible optimum; "best.found" for a feasible incumbent without a completed optimality proof; "infeasible" for a proved infeasible request; or "search.limit" when search or numerical-certification limits leave no certified feasible plan and infeasibility was not proved. This includes completed enumeration with unresolved boundary feasibility. The finite-normal joint produced-budget mode separately qualifies estimated quadrature bounds; numerical optimality does not constitute a rigorous interval certificate or validation of the biological normal approximation. Its additional statuses and cumulative-work diagnostics are described in st.cross.progeny.

`objective` Criterion, optimization direction, achieved value, relaxation bound, remaining gap, and bound description. A joint-mode bound is an upper bound on the selected mean or usefulness criterion; a fixed-mode bound is a lower bound on average within-cross relatedness.

`constraints` Requested family counts, progeny budget, relatedness limit, contribution bounds, effective integer use bounds, requested fixed contributions, and the number of allowed pairs.

`diagnostics` Constraint residuals, infeasibility or search-limit reason, node and proposal counts, accepted moves, starts, completed-search indicator, method used, relaxation bounds, estimated work, and whether a starting incumbent was used. The proposal count includes budgeted degree-realization attempts as well as local-change proposals. Boundary diagnostics include relatedness.roundoff, the roundoff half-width in the reported relatedness units, including numeric-conversion uncertainty; zero.row.residual, the largest row-relative residual for a zero limit; and uncertain.evaluations, the number of candidate evaluations with unresolved boundary feasibility. The reason "relatedness.numerical.limit" identifies unresolved relatedness certification.

`control` Search controls used for the request.

`search` Local seed, final random-number state, generator label and whether a supplied starting plan was used.

`inputs` The supplied merit vector, relationship matrix and allowed pairs, plus the optional score table, retained for reconstruction and review.

For the progeny argument, the returned plan identifies the solved count-dependent model, feasible incumbent, bound/gap and numerical/search qualification. Actual crosses$n.progeny and crosses$n.selected define its resources.

optimal qualifies the Burrows objective when its gap is closed; numerically.optimal additionally identifies the estimated numerical qualification of normal order statistics. A checked feasible result with an open or unknown gap has best.found status. Numerical boundary uncertainty can produce numerically.unresolved; absence of a feasible incumbent under search or allocation limits gives search.limit or resource.limit. Only a qualified impossibility result is infeasible. Shared cache, work, workspace and timing diagnostics are reported separately from the mathematical objective.

An object of class st.cross.frontier containing the requested limits and metric, point plans and statuses, summaries, starting provenance and shared resource diagnostics. Printing returns the input invisibly.

Joint progeny budgets

The named group has the following supported components:

`total.progeny` Required positive supported whole-number produced budget, at most the C integer maximum.

`n.selected` Positive whole-number scalar retained count per active family for the fixed-retention contract. Total retention is `n.crosses*n.selected`. It is mutually exclusive with the variable retained-budget components below.

`total.selected` Required positive supported whole-number retained budget for variable retention, at most `total.progeny`. Actual family counts sum to this budget.

`min.progeny` Common positive whole-number lower bound, defaulting to `n.selected` under fixed retention and to one under variable retention. Every actual produced count is at least its actual retained count.

`max.progeny` Common upper bound, defaulting to the total produced budget, no smaller than the lower bound.

`min.selected`, `max.selected` Common retained-count bounds for variable retention. The lower bound defaults to one and can explicitly be zero; the upper defaults to the smaller of the resolved common `max.progeny` and `total.selected`. Both are nonnegative whole numbers, with the upper no smaller than the lower. Actual retained counts are additionally bounded by actual production.

`bounds` Optional variable-retention data frame with character keys `P1Name`, `P2Name` and one or more of `min.progeny`, `max.progeny`, `min.selected`, `max.selected`. Each unordered biological family occurs at most once and belongs to the eligible graph. Common scalar defaults resolve first; provided columns override them for the keyed family. Missing rows and absent columns inherit those defaults. Missing cells, unknown columns and duplicate keys produce errors. Reciprocal directions share one biological count domain. These bounds require `total.selected`.

`selection.method` `"burrows"` by default, or `"order.statistics"`; the latter uses the qualified finite-normal order-statistic calculation of `st.cross.allocate`.

`control` Named allocation controls `max.cells` (default `10000000`), `max.transitions` (`100000000`), `tolerance` (`1e-10`) and `algorithm` (`"auto"`). Workspace includes shared caches and live inner-solver work. The transition allowance covers the whole joint call, rather than restarting for every proposed pair set. The group accepts `"auto"` or `"dp"`; `"greedy"` additionally requires `"order.statistics"`. Coupled practical count restrictions require `"auto"`. Variable retention supports `"auto"` and the compatible separable `"dp"` request; it does not support `"greedy"`. Unsupported combinations produce errors.

The top-level planner `control` still supplies its outer search method, seed, starts, proposals and node cap. The group is supported in joint mode. An explicitly supplied top-level `n.progeny` together with this group is an error; its default applies only to the equal-family path. Candidate scores require finite `mu` and nonnegative finite `va` for each allowed unordered pair. Stored `es` does not determine the count-dependent selection response.

With C active families, produced counts n_e, common retained count r, accuracy ρ and intensity $a(n_e, r)$,

$$\sum_e n_e = P, \qquad U_e = \mu_e + \rho\sqrt{v_e}\, a(n_e, r), \qquad J_{G1} = C^{-1} \sum_{e \text{ active}} U_e.$$

These formulas describe fixed common retention. With a total retained budget R, actual retained counts r_e instead satisfy

$$\sum_e r_e = R, \quad 0 \le r_e \le n_e, \qquad J_{G1} = \sum_{e:r_e>0} \frac{r_e}{R}\{\mu_e + \rho\sqrt{v_e}\, a(n_e, r_e)\}.$$

The corresponding mean criterion is $\sum_e (r_e/R)\mu_e$. Thus retention changes the weights of cross means as well as their selection intensities. Selection is still within families; a pooled top-R rule over all produced descendants is a different model. An inactive family produces and retains zero progeny. At most one eligible direction is selected for each unordered biological family. The usefulness criterion maximizes J_{G1}; the mean criterion maximizes the unselected family-mean baseline on the same retained weights.

Every active family produces at least one descendant and consumes one of the requested family slots. With an explicit zero retained lower bound, an active family can retain none. It still contributes to produced and family metrics, while its retained weight is zero. Its selected mean and intensity are unavailable; no intensity at zero retention is evaluated. Totals skip zero-weight rows explicitly. Full retention gives exactly zero selection response. The number of families contributing retained descendants can be smaller than the requested produced-family count.

Top-level parental contribution bounds, `theta` and family-slot diversity retain their family-slot basis. Practical restrictions additionally use their explicitly declared produced or retained bases. Original named `K`/`G` references and scales remain fixed. Unknown operation counts are not inferred from produced counts.

Exact small search enumerates biological pair/direction choices and solves the corresponding supported fixed-pair allocation, with valid bounds for the requested objective. Larger searches use bounded starts and exchanges. An exact inner allocation does not establish an outer joint optimum. An unfinished inner search leaves an unresolved branch unless an independent bound excludes it. Exhausting search or repair budgets does not prove infeasibility. A feasible supplied incumbent remains available when limits stop further improvement.

Variable-retention exact small solving uses both production and retention budgets. Larger problems retain checked incumbents with bounded count and family changes. The decreasing-produced-increment argument for fixed retention does not establish a joint optimum when retention changes. Unweighted family-average pruning does not bound the retained-weighted objective. A valid weighted relaxation or an unknown bound is reported. Original means are centered, with baseline and response compared separately in mean units, to preserve small responses under large common merit offsets.

The count-only intensity and discrete marginal cache is shared across the joint call. A change in parental merit or variance rescales the cached kernel rather than changing its mathematical definition. Numerical normal quadrature error estimates remain estimates rather than rigorous intervals. Complete integer search and biological prediction accuracy are distinct qualifications.

Produced-budget results have class `st.cross.plan`, with actual produced/retained counts, recomputed selected means and explicit method/basis provenance. `st.cross.evaluate` evaluates the same supplied population. A subsequent `st.cross.allocate` intentionally supplies new production bounds/budget for its fixed pairs, while retaining the source plan's operational and parental restrictions. It does not reimpose the source equal-size design used as a baseline.

Individual simulation commands accept the actual counts of varying families. The separate `st.cross.plan.execute` interface retains its equal-count execution contract.

A saved feasible plan is a warm start through `control$start`; current inputs independently rescore and check it. A warm restart is a new search and does not restore an unrecorded exact-search frontier. Variable-retention warm starts require actual `n.selected` counts; divisibility of the retained budget by the family count does not identify those counts. A fixed-retention plan can supply a start when its actual counts satisfy both new budgets and all current family bounds.

Gain-diversity frontiers

A common immutable genetic core, simulation banks and reusable caches support all frontier points. Genetic cell/work limits are cumulative across the whole call, including final assessments; forward simulation does not receive a new budget at each point.

Completed points reproduce standalone schedules with the recorded inputs, seed, start and bank when sufficient matching budgets are available. A point stopped by the shared budget is explicitly incomplete and need not match an uncapped standalone search. Infeasible, unavailable and unresolved outcomes remain in the requested sequence.

The frontier is a collection of assessed solutions, not a certificate between points. Heuristic or finite-bank points need not be monotone. Current family-slot quantities, approximate selected-founder proxies and actual current/future genomic metrics retain separate labels. Future assessment and paired uncertainty follow `st.cross.future`.

See Also

`propsd`, `st.ocs`, `st.cross.plan.execute`, `gs.cross.info`, `gs.cross.eval.mu`, `gs.cross.eval.va`, `gs.cross.eval.es`, `gs.estimate.gv`, `st.get.A`, `st.options`, `st.cross.progeny`, `st.cross.evaluate`, `st.cross.frontier`, `st.cross.future`

`st.cross.plan.execute` *Executes Named Families from a Cross Plan*

Description

Creates the specified families from exact named founders in an existing simulation population, with equal progeny counts and actual simulated individual identities and pedigrees.

Usage

```
st.cross.plan.execute(plan, population, name, scheme = "DH")
```

Arguments

`plan`	A result from `st.cross.plan` containing a nonempty `crosses` data frame. The columns `family`, `P1Name`, `P2Name`, and `n.progeny` specify the families independently of other result components. Family IDs are distinct positive whole numbers. The parent names identify individuals in `population`. Parent pairs are distinct, unordered and non-selfed. Progeny counts are equal positive whole numbers; family IDs and counts are numeric rather than factors.

`population` One character string naming an existing diploid founder population. Each selected founder ID occurs exactly once in this population. Selected founders are complete and homozygous at every installed simulation marker.

`name` One nonempty character string containing only ASCII letters and digits. This prefix determines the family populations, in plan-row order, as `name001`, `name002`, and so on. Suffixes have at least three digits. All target populations must be absent. The combined prefix and suffix leave space within the 255-character population-name limit for the internal F1 identity suffix.

`scheme` Either `"DH"` or `"F1"`. For `"DH"`, an F1 is generated from each planned pair and its meiotic products are doubled to obtain homozygous progeny. For `"F1"`, the offspring result directly from the planned parental crosses.

Details

Simulation initialization occurs before this function. `st.set.simpop` initializes the simulation state from a marker data set and creates the founder population. That initialization function performs its own reset; a single initialization can support several subsequent plans. The simulation random number generator is also initialized with `rng.init` before execution.

The installed simulation genome and marker map determine meiosis. The map contains markers on every chromosome, with valid chromosome lengths and positions. Existing crossover settings are retained. The initial execution scope is diploid line breeding with complete, homozygous selected founders. Missing marker states and heterozygous selected founders produce an error; execution does not impute or phase them. Valid chromosome segments flanking the installed markers remain supported.

Parent names select the actual founders used for each family. The function does not sample parental pairs from a larger pool. For DH production, one F1 from the inbred parents supplies the family; independent meioses from this F1 produce the requested doubled haploids. This corresponds to DH production without additional intermating generations.

All family counts, selected IDs, unordered pair uniqueness, supported founder states, and target-name collisions are checked before execution changes the simulation state. Counts and the total offspring-design size are bounded by `.Machine$integer.max`, even when a planner result permits a larger produced-progeny budget. Errors or interruptions during execution remove all populations newly created by that call, including completed families from the same call. Existing founder and unrelated populations are preserved. Simulation random-number and identity streams may advance during an interrupted or failed execution; their earlier states are not restored.

The function preserves the existing genome, map, effects and thread settings. It does not reset simulation state or change the caller's R random-number state. The returned offspring names and pedigrees are generated by the simulation engine from the actual parents and F1, rather than assigned from an intended pairing after a random cross.

Value

An invisible ordinary data frame with one row per family and columns `family`, `population`, `P1Name`, `P2Name`, `n.progeny`, and `scheme`. Its ordinary printed display contains these compact family summaries.

The `design` attribute is a data frame with one row per simulated individual. Its columns are `family`, `population`, `name`, `P1Name`, `P2Name`, `F1`, `pedigree`, and `scheme`. `name` and `pedigree` contain

the actual offspring identity and pedigree. F1 contains the actual intermediate F1 identity for DH families and NA for direct F1 production. The family populations remain available to the other simulation functions.

References

Maurer HP, Melchinger AE, Frisch M (2008) Population genetic simulation and data analysis with Plabsoft. Euphytica 161:133–139.

See Also

st.cross.plan, st.set.simpop, rng.init, population.pedigree, genotype.population, evaluate.population

---

st.data.combine *Combine Individuals from Marker Data Sets*

Description

Creates a marker data set by appending individuals from source data sets having exactly the same marker identifiers. Validation, matching, copying and storage are implemented in C.

Usage

```
st.data.combine(target.data.set, source.data.sets)
```

Arguments

target.data.set
: One nonempty character string naming the target marker data set. It must differ from every source name.

source.data.sets
: A nonempty list of single dataset-name strings, or a character vector of dataset names. All sources must already contain valid marker data.

Details

Every source must have exactly the same set of unique marker identifiers. Marker order may differ: matching uses marker names, and the output marker order follows the first source. No intersection, union, automatic dropping of markers or allele recoding is performed. The observed allele sets may differ between sources, including fixed alleles and missing observations. Allele codes must already be compatible. For nucleotide VCF inputs, strictACTG=TRUE in read.vcf or st.load.vcf.data provides fixed codes independent of REF/ALT order.

Individuals appear in source-list order and retain their order within each source. Individual identifiers must be unique across the combined data; duplicate identifiers, repeated source names and missing source names cause an error. Names and pedigrees are copied independently of the sources. Genotype order and missing values are preserved. The output is marked as phased only if every source is marked as phased.

The linkage map is copied from the first source, including its map order and mapping to marker rows. Maps from other sources are not combined or compared. If the first source has no linkage map, neither does the output.

Available phenotypes are appended by individual. If only some sources contain phenotypes, individuals from the other sources receive missing phenotypes. Marker statistics are recalculated, and the usual automatic packed-storage policy applies. Fitted effects, prediction state, relationship matrices and derived workspaces are not inherited.

Source data sets are unchanged. A pre-existing target is replaced only after the combined data have been prepared successfully. A target referenced by a hybrid data set cannot be replaced. Invalid input leaves an existing target unchanged and does not create a new empty target.

Large genotype copies use the thread budget set by `st.set.num.threads` when the package has OpenMP support.

Value

Integer zero, invisibly, on success. Invalid or incompatible inputs cause an error. The combined marker data set is stored as a side effect.

`st.data.set.list` *List SelectionTools Marker Data Sets*

Description

Returns a compact summary of the marker data sets currently supported 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. `FALSE` (the default) prints only the compact overview columns `data.set`, `no.markers`, `no.individuals`, `map`, `BitsPerMarker`, `A_matrix`, `Z_matrix`, `ram.core.MB`, and `ram.work.MB`. `TRUE` prints 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 homologue 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.

`A_matrix` indicates whether a valid genomic relationship matrix is cached; its validity is checked before reporting the flag. `Z_matrix` indicates whether the marker design matrix Z is currently stored. Reporting these flags does not construct either matrix. Both matrices belong to the work portion of the data set.

`BitsPerMarker` is the allele-payload width per diploid marker call (per marker and individual): 4, 8, or 16 bits. It replaces the former `bits.per.genotype` column. Dictionary storage is additional.

`ram.core.MB` and `ram.work.MB` are estimates in MiB (1024^2 bytes), with the unit shown as `MB` in the column names. `ram.core.MB` includes individual identifiers and full pedigrees, marker names, allele payload and dictionaries, linkage-map storage, observed responses and the active symbolic-A fitted prediction state. Shared fitted state is counted in each owning dataset's estimate. `ram.work.MB` contains reconstructible calculation state, including marker statistics, earlier genotypic values and effects, 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`, `BitsPerMarker`, `A_matrix`, `Z_matrix`, `ram.core.MB`, and `ram.work.MB`. `print(x,complete=TRUE)` displays 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 supported data set and the following columns. If all supported 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.
- A_matrix: whether a valid genomic relationship matrix is cached. Cache validity is checked before reporting this flag; cached numeric storage belongs to ram.work.MB.
- BitsPerMarker: current allele-payload width, 4, 8, or 16 bits per diploid marker call (per marker and individual); dictionary storage is additional.
- Z_matrix: whether the marker design matrix Z is currently stored.
- ram.core.MB: estimated marker-data and linkage-map RAM in MiB.
- ram.work.MB: estimated calculated/result/workspace RAM in MiB.

The digits setting is stored as an attribute and changes printing only.

---

st.data.set.remove *Removes a SelectionTools Marker Data Set*

Description

Releases all compiled 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 compiled 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 supported.

The function does not create a missing data set. If data.set is not supported, 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 supported.

st.data.set.remove.all

Removes 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 compiled storage belonging to supported 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 supported.

Value

Invisibly returns the number of user-defined data sets removed. The built-in `"default"` data set remains supported 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

Constructs a path in the SelectionTools data directory

Description

Constructs 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` *Converts 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 optional `allele.codes` attribute is a list of integer coding vectors, named and ordered by marker ID. It records the internal allele codes corresponding to REF and successive ALT entries. STvcf exchange retains this attribute so fixed or subsetted loci keep the same coding as their fitted marker reference. The loader validates it before replacing a destination. Conversions preserve it; a manually edited marker list must keep it aligned. It is not part of plain-text VCF exchange.

The optional versioned `pedigree` attribute is a data frame with `name` and `pedigree` columns and integer attribute `version=1L`. Its names must match the individual identifiers in order. Without it, individuals are founders. Full strings are owned independently of ancestor populations. The optional `prediction.state` attribute preserves a compatible active symbolic-A fitted state during STvcf/RDS exchange. Conversion in either direction retains both attributes; loading validates them before replacing the destination.

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 identifiers must be non-empty and contain fewer than 256 bytes. Individual identifiers must be non-empty; they use dynamically allocated strings subject to the R string-size limit. Whitespace and control characters are rejected; ordinary punctuation such as dots, underscores, and hyphens is permitted. 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 identifier 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 *Constructs a path in the SelectionTools data directory*

Description

Constructs 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` *Defines 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. The result is returned directly from memory; this argument does not name an output file.
`auxfiles`	Logical. If `TRUE`, the result is returned directly from memory.
`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 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 *Randomizes 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. A call to `set.seed()` before `st.destroy.phase()` provides reproducible randomized phase. The marker data set is modified in place and its compiled 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.detect.core *Detect a Haplotype Core Collection*

Description

Selects compact core collections from one phased marker data set using either forward selection or exact best-subset selection on distinct block-specific haplotypes.

Usage

```
st.detect.core(pool, n.max,
               haplotype.options = NULL, blocks = NULL,
               exclude.compatible = TRUE, method = "forward")
```

Arguments

`pool`	Name of the phased marker data set containing the candidate genotypes.
`n.max`	Positive integer maximum number of genotypes in the core.
`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. Exactly one of `haplotype.options` and `blocks` is required. An empty list requests the default five-marker 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 contain consecutive markers on one chromosome.

`exclude.compatible`
: Logical. `TRUE` selects the conservative missing-compatible rule. In forward selection, a complete haplotype is not counted as a marginal gain when it is compatible with an incomplete haplotype already present in the selected core. In best-subset selection, a represented complete haplotype is excluded from the optimization score when it is compatible with an incomplete haplotype anywhere in that candidate subset.

`method`
: Selection method. `"forward"` adds one genotype at a time using the largest current marginal gain. `"best.subset"` finds the exact optimal subset separately for each core size up to `n.max`.

Details

The marker data set must already have consistent gametic phase. Phasing is not performed by this function. Haplotype identity is the ordered sequence of marker allele states within a block. The same sequence in two different blocks counts as two block-specific haplotypes. Only complete haplotype sequences are counted as represented segments.

With `method="forward"`, selection starts with an empty core. The first genotype is the one representing the largest number of distinct complete block-specific haplotypes. At every subsequent step, the remaining genotype contributing the largest number of previously unrepresented qualifying haplotypes is added. Shared haplotypes count once. Equal gains are resolved by bytewise lexicographic genotype name.

With `method="best.subset"`, an exact branch-and-bound search is carried out separately for core sizes 1 through `n.max`. For each size, the subset maximizing the number of qualifying represented haplotypes is returned. When `exclude.compatible = TRUE`, a complete variant is excluded from this objective if it is compatible with an incomplete haplotype in the candidate subset. Branch-and-bound bounds ignore future compatibility masking only when forming an upper bound; because additional masking can only lower a descendant score, this pruning remains exact. Equal optimal subsets are resolved by the first combination of genotype names in bytewise lexicographic order. Runtime can grow combinatorially for large pools or large values of `n.max`; no heuristic result is silently substituted.

With `exclude.compatible = TRUE`, incomplete haplotypes in the already selected core can make otherwise new complete variants compatible with the current core. Such variants are excluded from the marginal selection gain. The `segments` values nevertheless report the cumulative number of distinct complete block-specific haplotypes actually represented by the selected genotypes.

Forward selection stops when `n.max` genotypes have been selected or when no remaining genotype has a positive marginal gain. Exact best-subset search stops early once an optimal subset represents every complete block-specific haplotype observed in the pool. Thus either method can return fewer than `n.max` core sizes.

Counting and both search methods are implemented in C. OpenMP builds use the existing `st.set.num.threads` setting. Exact search parallelizes independent first-genotype branches; forward search parallelizes candidate scoring. Results, including ties, are independent of thread count. Input marker data and block definitions are not modified, and no random numbers are consumed.

Value

A list with two elements:

`genotypes` A list of core sets. With forward selection, element `k` contains all genotype names selected through step `k`. With best-subset selection, element `k` is the exact optimal subset of size `k`; its names are in bytewise lexicographic order.

`segments` An integer vector parallel to `genotypes` giving the cumulative number of distinct complete block-specific haplotypes represented by each core set.

If no complete haplotype can be represented, both elements have length zero.

See Also

`st.select.enrichment`, `st.calculate.enrichment`, `st.def.hblocks`, `st.phase`

`st.diallel.cross` *Identify Unordered Diallel Crosses*

Description

Constructs a common cross factor for the two parental roles in a half diallel.

Usage

```
st.diallel.cross(mother, father)
```

Arguments

`mother`, `father`	Equal-length vectors of parent identifiers. Factors are converted to their labels. Missing either parent produces a missing cross.

Details

Swapping the parental roles leaves the cross identifier unchanged. Length-prefixed labels avoid collisions when parent names contain the separator. The labels are identifiers, not pedigree expressions.

`random=~overlay(mother, father)` in `st.lmm` specifies general combining ability. The cross factor enters as a separate random term for specific combining ability. The overlay uses one shared parental effect vector; each parent has loading one and a self cross has loading two. Reciprocal and maternal effects must be specified separately when required.

Value

A factor of canonical unordered parent pairs, with missing values where either parent identifier is missing.

See Also

`st.lmm`, `st.predict`

st.genetic.distances *Genetic Distances and Population Plots*

Description

The function *st.genetic.distances* calculates genetic distances for each possible pair of individuals. The plotting helpers draw a genetic-distance heatmap and principal-coordinate score or variance plots using standard R graphics.

Usage

```
st.genetic.distances(measure = "mrd", format = "l",
                     filename = NULL, auxfiles = FALSE,
                     data.set = "default")
st.plot.genetic.distances(x, labels = FALSE, cluster = TRUE,
                          col = grDevices::heat.colors(200),
                          hclust.method = "complete", ...)
st.plot.pcoa(x, which = c("scores", "variance", "cumulative"),
             axes = c(1L, 2L), groups = NULL, col = NULL,
             pch = NULL, legend = TRUE, cex = 0.8, ...)
```

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. The value "l" specifies 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.
x	For the distance heatmap, a dist object or a finite, nonnegative symmetric distance matrix with zero diagonal. For the principal-coordinate plots, a stats::cmdscale result calculated with eig = TRUE, containing points and eig.
labels	TRUE shows individual names in the distance heatmap; FALSE omits them. A character vector supplies one label per individual.
cluster	TRUE clusters the distance heatmap; FALSE retains input order.
col	Heatmap colours, group colours for coordinate scores, or the colour of variance-plot points. Group colours are recycled as needed.
hclust.method	Linkage method passed to stats::hclust. Clustering uses the supplied pairwise distances directly.

`which`	Coordinate scores, individual percentages of explained variance, or cumulative percentages of explained variance.
`axes`	Two different principal-coordinate indices for a score plot.
`groups`	NULL, or one nonmissing group per individual. If this vector has names and the coordinates have row names, groups are aligned to individuals by name. Otherwise their order must agree.
`pch`	Symbols for the groups in a score plot, recycled as needed, or the symbol for a variance plot.
`legend`	TRUE adds a group legend to a score plot when groups are supplied.
`cex`	Point size for principal-coordinate plots.
`...`	Named arguments passed to `stats::heatmap` or `graphics::plot`, such as margins, a title or axis labels. Data, distance scaling, clustering and group-aesthetic arguments must be supplied through the explicit arguments above.

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 $\sqrt{2m}$, 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.

The distance heatmap uses the same ordering on both axes and never scales rows or columns. With clustering enabled, hierarchical clustering is based on the original pairwise genetic distances, not distances between rows of the distance matrix. Its dendrogram can therefore differ from an unmodified call to `heatmap(as.matrix(x))`.

The principal-coordinate helper draws already calculated coordinates and does not repeat the distance calculation or ordination. Explained-variance percentages divide each positive eigenvalue by the sum of positive eigenvalues; negative eigenvalues are assigned zero for these display percentages. For a non-Euclidean distance this convention does not measure variation associated with negative axes. These helpers open no graphics device and add no package dependencies.

Value

`st.genetic.distances` 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.plot.genetic.distances` invisibly returns the `heatmap` result, including row and column order indices. `st.plot.pcoa` invisibly returns a list with the supplied `points`, the displayed `variance.percent`, and the aligned `groups` when a score plot is drawn.

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

Calculates genetic distances from SelectionTools marker data

Description

Calculates 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

Calculates genetic distances from SelectionTools marker data

Description

Calculates 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.A — *Constructs a genomic relationship matrix from a SelectionTools data set*

Description

Obtains the named marker design matrix from a SelectionTools marker data set and applies the existing VanRaden relationship matrix constructor. Both operations are called from C.

Usage

```
st.get.A(data.set = "default", coding = "0 1 2")
```

Arguments

data.set	Name of an existing SelectionTools marker data set, loaded for example with st.read.marker.data.df or st.load.vcf.data.
coding	Coding of the internal marker design matrix; passed unchanged to gs.make.A. The default matches the standard internal 0/1/2 coding.

Details

The compiled wrapper calls the existing named Z accessor, which builds Z from the marker alleles if necessary, followed by the existing A constructor. It does not implement a second marker converter or relationship formula. For polymorphic biallelic input, the result is equivalent to gs.make.A(gs.get.Z(data.set, names=TRUE), coding=coding). This relationship-matrix path also accepts fixed biallelic loci, which contribute zero after centering. An entirely fixed panel has zero relationship scale and is rejected. Multiallelic loci must be excluded. The separate older gs.get.Z interface retains its

original marker requirements. The dedicated genomic_relationship_A field caches the numeric relationship matrix. The cache is owned by the compiled marker dataset in C memory and is independent of returned R matrices. Removing or modifying an R result does not release or alter this cache. compiled symbolic fits reuse it across calls. Its validity is checked against marker content, individual and marker names and order, and coding; performance-only changes do not invalidate it. Compaction clears the cache. This exported accessor returns a copy, so editing the returned matrix cannot change the internal cache. Fitted objects retain independent model snapshots; removing the source dataset or clearing its cache does not invalidate a fit. No ridge or inverse is applied.

Value

A numeric square relationship matrix. Individual identifiers label both dimensions. Invalid inputs or construction failures produce an error.

See Also

gs.get.Z, gs.make.A, st.read.marker.data.df, st.load.vcf.data

Examples

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
A <- st.get.A("maize.example")
A[1:4, 1:4]
```

---

st.get.info.level *Returns the current SelectionTools information level*

Description

Returns the current SelectionTools information level.

Usage

```
st.get.info.level()
```

Value

An integer scalar giving the current SelectionTools information level.

st.get.map *Returns the linkage map for a SelectionTools data set*

Description

Returns the genetic map, or the stored physical map when no genetic map is present, for a SelectionTools data set.

Usage

```
st.get.map(filename = "map", data.set = "default")
```

Arguments

filename	Character string retained for compatibility. The map is returned directly from memory; this argument does not create a file.
data.set	Name of the SelectionTools data set.

Details

A physical map imported from STvcf is stored independently of genetic positions. When it is returned, Class = "bp" identifies the base-pair units of Pos. Genetic-map returns retain their previous interpretation. Only markers currently present in the dataset are returned.

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 *Reports the computational thread configuration*

Description

Returns the package thread budget, or report the package, OpenMP and BLAS configuration without launching an OpenMP team.

Usage

```
st.get.num.threads(complete = FALSE)
```

Arguments

complete	If FALSE, return the package thread budget as one integer. If TRUE, return the detailed configuration as a named list.

Details

The default result is the stored budget for SelectionTools OpenMP kernels; it is not a measurement of an active team. It is 1 in a build without OpenMP. Runtime and backend limits and operation size may reduce actual team sizes. The startup hook calls `st.set.num.threads(2)`. An explicit setter call caps the effective OpenMP budget at the runtime thread limit, disables dynamic team shrinking, and applies the effective budget to supported BLAS controls. A build without OpenMP limits an OpenMP OpenBLAS, or FlexiBLAS with an unknown backend threading model, to one BLAS thread.

BLAS detection uses the library supplying R's linked BLAS operations. If that library does not expose a supported control interface, the reported BLAS count is `NA` and `blas.control` is `FALSE`. This includes the serial reference BLAS supplied with R: the absence of a query interface is not treated as evidence that an unidentified BLAS is serial.

Value

With `complete = FALSE`, an integer scalar between 1 and 256. With `complete = TRUE`, a named list containing:

requested The initial budget or most recent explicit setting, after clamping to the range 1 to 256.

package.threads The effective budget used by package OpenMP kernels, capped by the runtime thread limit, or 1 without OpenMP.

explicit Whether a setter has been called in this process.

openmp.available Whether SelectionTools was built with OpenMP.

openmp.max.threads The OpenMP maximum for the calling thread, or 1 without OpenMP.

openmp.dynamic Whether the OpenMP runtime may adjust team sizes, or `FALSE` without OpenMP.

openmp.thread.limit The runtime's limit on OpenMP threads, or 1 without OpenMP.

blas.backend The identified interface: `"OpenBLAS"`, `"FlexiBLAS"`, `"MKL"`, or `"unavailable"`.

blas.library The path of the library supplying the linked BLAS, when available.

blas.threads The configured thread count reported by the backend, or `NA` when it cannot be queried. It is not a measurement of the threads used by a particular BLAS operation.

blas.control Whether the supported backend's thread setter and getter are available.

blas.parallel The OpenBLAS threading mode: 0 for a serial build, 1 for platform threads, or 2 for OpenMP; `NA` when unavailable.

legacy.outer.threads The thread count for older genomic-selection outer loops that call BLAS. These loops use 1 to avoid unsafe concurrent BLAS calls; BLAS itself retains its configured budget.

See Also

`st.set.num.threads`

`st.get.simpop` *Returns 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 follow the same identifier rules as the VCF interface: names must be non-empty, occupy fewer than 256 bytes, and contain no spaces or ASCII control characters. Punctuation such as hyphens, dots, and underscores is preserved; 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 retains each individual's stable identifier and complete stored simulated pedigree. Imported founders retain their imported identifiers. New cross, DH and SSD individuals retain their creation names, including their full pedigree expressions. Copies and population renames keep these identities. A simulation population containing repeated copies with the same identifier cannot be exported to a marker dataset; the destination is left intact. `population.pedigree` reports names and pedigrees in either representation.

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 compiled 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

Returns 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

Returns 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 supports only diploid simulation data. Stable individual names and full simulated pedigrees are retained, using the same rules as `st.get.simpop()`. Duplicated individual IDs are rejected before replacing the destination. Population names must contain only ASCII letters and digits. Marker names follow the same identifier rules as the VCF interface: names must be non-empty, occupy fewer than 256 bytes, and contain no spaces or ASCII control characters. Punctuation such as hyphens, dots, and underscores is preserved. 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 compiled 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.gwas *Single-Marker Association Scans with Mixed Models*

Description

Tests additive marker effects with optional genomic relationships, population covariates and known residual covariance. Optionally calibrates a genome-wide significance threshold by parametric simulation from the fitted null model.

Usage

```
st.gwas(fixed, random, data, marker.data.set = "default",
        K = NULL, n.PC = 0L, min.MAF = 0.05, P3D = TRUE,
        max.missing = 1, missing = "omit", map = NULL, rcov = NULL,
        control = list(), sim.threshold = list())
compare.gwas.fits(..., adjust = "FDR", alpha = 0.05, trait = NULL)
summarize.gwas.fit(fit, adjust = "FDR", alpha = 0.05,
                   top = 6L, trait = NULL)
```

```
summarize.gwas.simulation(fit, alpha = 0.05, top = 6L, trait = NULL)
## S3 method for class 'st.gwas.summary'
print(x, digits = NULL, char.per.line = NULL, ...)
## S3 method for class 'st.gwas'
as.data.frame(x, ...)
## S3 method for class 'st.gwas'
print(x, digits = NULL, top = NULL, bottom = NULL,
                     char.per.line = NULL, ...)
## S3 method for class 'st.gwas'
plot(x, which = "manhattan", trait = NULL,
                    alpha = 0.05, adjust = "BH", threshold = NULL,
                    chromosome = NULL, xlim = NULL,
                    position.scale = 1e6, position.unit = "Mb",
                    shade = NULL, highlight = NULL, labels = FALSE,
                    level = NULL, main = NULL,
                    col = c("grey35", "cornflowerblue"),
                    cex = 0.5, pch = 16, ...)
```

Arguments

`fixed`	Two-sided fixed-effects formula, including numeric covariates and factors as needed. A `cbind` response requests separate univariate scans for its columns, not a joint multivariate test.
`random`	Required one-sided formula identifying the genotype column, for example `random = ~gen`. Exactly one untransformed column name is supported. The polygenic effects for these genotype levels have covariance proportional to `K`; `K = "A"` constructs that relationship internally from the marker panel. The column contains nonmissing marker-individual identifiers. Repeated records for a genotype share its polygenic effect. With `K = FALSE`, the polygenic effect is omitted, but the formula still identifies marker individuals.
`data`	Phenotype data frame. Repeated records per individual are allowed. Column names beginning with `.st.gwas.` are reserved.
`marker.data.set`	compiled marker dataset name, or numeric matrix with individuals in rows and markers in columns. Matrix row and column names must be unique identifiers. Values are allele dosages between zero and two, including fractional imputed dosages; `NA` denotes missing calls. Converts rrBLUP scores in minus one, zero, one coding by adding one. compiled datasets are read directly without exporting the complete matrix.
`K`	NULL (the default), or FALSE, omits kinship adjustment. The required `random` formula still identifies marker individuals; no polygenic random effect is fitted when kinship is omitted. The quoted string `"A"` builds a VanRaden relationship internally from `marker.data.set`, using all polymorphic biallelic markers with mean imputation. Symbolic `K = "A"` does not evaluate a caller object named `A`. Unquoted `K=A` evaluates the actual R object `A`. An alternative is a named finite symmetric positive-semidefinite matrix covering every marker individual. Its axes are aligned by identifier. Kinship accumulation, eigendecomposition

and PC calculation use compiled buffers. Exported copies are retained in the returned null models for compatibility.

`n.PC` Number of leading eigenvectors of `K` added as fixed population covariates. Requires explicit `K = "A"` or a supplied relationship matrix. For an ordinary scan with supplied population covariates, `fixed` contains those covariates and `K=FALSE,n.PC=0` specifies the scan.

`min.MAF` Minimum minor allele frequency, between zero and one half. Frequency is computed over unique phenotyped individuals retained for the trait and available fixed predictors, not over replicated records.

`P3D` TRUE fits the null covariance once and holds its shape fixed during the scan. FALSE re-estimates free covariance parameters per marker. The same setting is used for every simulated scan. Each simulated response receives its own null-model fit with `method = "parametric"`. `method = "parametricFixedVC"` requires P3D = TRUE and reuses the original null covariance across replicates.

`max.missing` Maximum missing genotype fraction per marker, over unique retained phenotyped individuals.

`missing` `"omit"` removes all records of individuals with a missing call at the tested marker. `"mean"` explicitly imputes their dosage by the observed mean for retained individuals.

`map` NULL, or a data frame with `marker`, `chromosome` and `position` columns matched by marker identifier. A compiled dataset supplies its stored physical map by default, falling back to its genetic map when no physical map is stored. Physical coordinates imported from the STvcf `physical.map` attribute need no separate `map` argument. Unmapped markers use scan order in plots. Positions retain the units of the supplied map.

`rcov` Optional known residual covariance matrix, on its absolute response scale. Naming, alignment and restrictions are those of `st.lmm`. No additional residual variance is fitted.

`control` Fitting controls passed to the compiled model. Additionally `batch.size` sets the number of marker columns processed together (default 64; from 1 to 4096).

`sim.threshold` A list controlling simulated significance thresholds. The default empty list disables simulation and does not use the random number generator. A nonempty list enables simulation and accepts only the uniquely named entries `nsim`, `alpha`, `seed`, and `method`. The method is either `"parametric"` (the default) or `"parametricFixedVC"`. Omitted entries default to `nsim = 999L`, `alpha = 0.05`, and `seed = NULL`. `nsim` must be a positive integer and `alpha` must be a finite scalar strictly between zero and one. The number of replicates must satisfy `floor(alpha * (nsim + 1)) >= 1`. `nsim` must be smaller than `.Machine$integer.max`, and a non-NULL `seed` must be a finite integer with absolute value no larger than `.Machine$integer.max`. An explicit seed restores the caller's previous random-number-generator kind and state, including the absence of `.Random.seed`. With NULL, one master seed is drawn from the current generator and recorded; only that draw advances the caller's state. Scoped simulation does not support Box–Muller or user-supplied random-number generators. Unknown, duplicated, or unnamed list entries are errors.

`fit` One fitted `st.gwas` object to summarize.

`x` An `st.gwas` object, or a `st.gwas.summary` result for its print method.

`digits, top, bottom, char.per.line`
For summaries, `top` is a positive integer number of leading markers, supplemented by any significant chromosome leads outside this set. Otherwise these are compiled table printing controls; row limits may be `Inf`.

`which` One or more of `"manhattan"`, `"qq"`, and `"region"`, in the requested order. Plotting does not change the layout of the graphics device. A regional plot requires `chromosome`.

`trait` NULL for all traits, or one trait name to plot or summarize.

`alpha` Significance level for summary decisions or a BH, Bonferroni, or unadjusted plotted threshold. When plotting a simulated threshold, an omitted `alpha` uses its stored level. An explicitly supplied different level is an error; recalibration is requested through `sim.threshold` in `st.gwas`.

`adjust` For summaries, `"FDR"` (the default), its alias `"BH"`, `"none"`, or `"simulation"`. FDR uses the stored trait-wise Benjamini–Hochberg probabilities. Summary output has no Bonferroni column. For plotting: `"BH"`, `"bonferroni"`, `"none"`, or `"simulation"` for Manhattan and regional thresholds. When this argument is omitted and simulation results are present, the stored simulated threshold is selected automatically. Without simulation results, the default is BH. Supplying an adjustment explicitly overrides this automatic selection. BH is the Benjamini–Hochberg procedure; rrBLUP plots use Storey q-values. An explicit `"simulation"` request requires a complete simulated threshold for each requested trait.

`threshold` NULL uses the automatic or explicitly selected adjustment; FALSE omits the horizontal significance line. Alternatively, one or more raw p-values in (0, 1] specify the drawn thresholds. FALSE and numeric values override `adjust`, including automatic simulated thresholds. The stored simulated cutoff is drawn directly on its minus-log10 scale, so its raw p-value may underflow to zero without losing the line. Plotting never runs or repeats simulations.

`chromosome` One chromosome identifier for a regional plot.

`xlim` Two limits for a regional plot, in the original units of the supplied map. The threshold is still computed from the complete trait scan.

`position.scale` Positive divisor converting map positions to regional plot coordinates. The default `1e6` converts base pairs to megabases.

`position.unit` Unit label for the regional horizontal axis. This label and `position.scale` must agree with the supplied map.

`shade` NULL, or two positions in the original map units delimiting a shaded interval in a regional plot.

`highlight` NULL, or marker identifiers to highlight in coral.

`labels` TRUE labels the highlighted markers; FALSE omits labels. A character vector labels the specified marker identifiers.

`level` NULL omits a QQ envelope. A probability between zero and one adds a pointwise beta-order-statistic reference envelope. This envelope assumes independent uniform null p-values; marker linkage disequilibrium means that it is not a simultaneous genome-wide confidence band.

main	NULL for the trait name, or a plot title.
col	Plot colours. A Manhattan plot cycles these colours by chromosome; QQ and regional plots use the first colour.
cex, pch	Point size and plotting symbol for customized plots.
...	For compare.gwas.fits, individual fitted st.gwas objects. Named arguments supply model labels; otherwise bare fitted-object names are used, with fit1, fit2, etc. as fallback labels for expressions. Model labels must be unique; name repeated inputs explicitly. For customized plots, named graphical arguments passed to graphics::plot, such as xlab, ylab, ylim, and bty; x, y, and type are reserved. Reserved for the printing and data-frame methods.

Details

Without rcov, tests use an F statistic with numerator df one and denominator df equal to the marker model residual df. With P3D the null covariance shape is retained and the marker model residual scale is re-estimated. This follows the testing convention of rrBLUP's GWAS function for complete biallelic genotypes. Known absolute rcov instead gives a one-df GLS Wald chi-squared test, without rescaling the entire covariance matrix.

For omitted genotypes, the covariance is subset to retained observations and solved again. A principal submatrix of the original inverse is not substituted for this calculation. Thus missing-call results can differ from rrBLUP's P3D shortcut. Fixed-effect rank and estimability are checked for each tested marker.

Multiallelic, monomorphic, missing, filtered or nonestimable loci have a status and no association statistic. Marker-specific fitting failures produce a message and allow the remaining scan to continue. compiled reference/effect allele values are stored biological allele codes, not necessarily nucleotide strings. With strictACTG input they use A=1, C=2, G=3, T=4. Numeric-matrix effects are per unit of supplied dosage.

BH and Bonferroni adjustments are computed separately per trait over successful tests. Bounded batches avoid an additional complete dosage export, but fitting still requires the dense relationship/covariance storage used by the selected solver. The global thread setting from st.set.num.threads is respected and is not changed. The scan does not change marker data or stored prediction state. st.gwas returns its result without opening a plot; plotting is a separate call to plot.

Simulated significance thresholds: With a nonempty sim.threshold list, the original marker-free null model generates independent Gaussian response replicates. Its fitted fixed mean and covariance parameters define the generating model. Random effects and errors are regenerated jointly, using the unconditional simulation convention re.form = NA of simulate.st.lmm. The realized fitted random effects are not held fixed.

The marker data, resolved relationship matrix, principal components, fixed-effects design, phenotype-to-individual mapping, and marker filters remain those of the original analysis. Original missing response cells are restored before each scan. Responses expressed through a formula transformation are simulated on the retained analysis scale, with no second application of that transformation. Repeated observations retain their common individual mapping and its covariance structure. Known absolute rcov retains the GLS Wald testing convention described above; otherwise the simulated scans use the same F tests as the original.

With method = "parametric", each replicate refits its null model. With P3D = TRUE, its fitted covariance shape is reused over that replicate's markers and each marker retains its residual-scale estimate. With P3D = FALSE, free covariance parameters are refitted per marker. Missing

genotype calls are handled by the original `missing` setting, including the covariance solve on retained observations for omitted observations.

With `method = "parametricFixedVC"`, the original fitted variance components are held fixed for every replicate and marker; no simulated model undergoes variance-component estimation. This method requires `P3D = TRUE`. The null covariance factorization and complete or mean-imputed marker information are reused. Fixed effects are projected out separately for each response, and each marker retains its response-specific residual-scale estimate for F tests. Omitted genotype calls use a covariance solve on retained observations with the same original variance components. This is a faster conditional, plug-in calibration; it does not reproduce the variation from re-estimating variance components. Both methods generate Gaussian responses from the same original fitted null model and use the same marker family and Monte Carlo calculation.

For each trait, the eligible marker family is fixed by the original analysis and the maximum compiled minus-log10 p-value is recorded for every simulated genome scan. Here B is `nsim`, M_b denotes these maxima, and s is an observed marker score. The simulation-adjusted p-value is

$$p_{\mathrm{sim}}(s) = \frac{1 + \sum_{b=1}^{B} I(M_b \geq s)}{B + 1}.$$

The comparison includes ties. The definitions are $h = \lfloor \alpha(B+1) \rfloor$ and $k = B + 1 - h$. The score cutoff is the kth ordered maximum in ascending order; scores strictly above it meet the requested level. Significance decisions use `p.sim <= alpha`. The cutoff is an observed order statistic, without interpolating between maxima. Its raw p-value is supplied for convenience, but the score remains authoritative if the raw value underflows.

The nominal level concerns the maximum over markers within one trait under its fitted global null model. Fitted generating parameters and a finite simulation count make this a model-based approximation. Every trait is calibrated separately; the level does not jointly cover several traits or different model choices.

Failures of eligible marker tests or replicate fits, including nonfinite simulated scores, are retained in the diagnostics. Failed draws are not replaced, and failed markers are not silently removed from the simulated maximum. If a trait's calibration is incomplete, its threshold and simulation-adjusted p-values are unavailable. The maxima, settings, and failure diagnostics remain in the returned object. A trait with no eligible markers has no simulated threshold. Only replicate maxima are retained, not every simulated marker table.

The existing Manhattan and QQ plotting calls are retained. Customized Manhattan, QQ and regional plots use standard R graphics without additional packages. A mapped Manhattan plot orders numeric chromosome labels numerically and keeps the encounter order of other chromosome labels. If any successful marker lacks a map position or chromosome, it uses marker order instead. Regional plots require mapped markers. The BH line identifies the largest raw p-value among BH discoveries, if any; it represents false-discovery-rate control and is not a familywise threshold. The Bonferroni line uses the number of successful tests in the complete trait scan, even when plotting a region.

A completed simulated threshold is used automatically for Manhattan and regional plots unless `adjust` or `threshold` supplies an override. It is shown as a coral dashed line labelled with the simulation level and calculated score cutoff. The automatic vertical range includes that line. Regional significance colouring uses the simulation-adjusted p-values and the stored level. If automatic selection encounters an incomplete calibration, a warning is given and the significance line is omitted. An explicit `adjust = "simulation"` request instead fails with an error. Neither route substitutes a BH threshold for an incomplete simulated threshold.

For customized plots, exact zero raw p-values are clipped to the smallest positive normal double when drawing QQ points, explicitly supplied raw-p-value thresholds, or association points with nonfinite scores. Finite compiled association scores and stored simulated score cutoffs are used directly.

Value

The three summary functions return a serializable `st.gwas.summary` list with `models` (model, trait, PCs, tested, significant), `markers` (model, trait, chromosome, marker, position, MAF, effect, p, p.adjusted, score), `adjust`, `alpha` and `comparison`. Printed adjusted probabilities are labelled FDR or p.sim as appropriate. Printing uses the common compiled table printer, respects `char.per.line` and prints all returned rows. Probabilities retain `digits` decimal places, with values below the display limit shown as, for example, `<0.0001` at four decimal places.

A serializable `st.gwas` list containing `results`, `null.models`, trait labels and scan settings. The results table has `trait`, `marker`, `chromosome`, `position`, `reference.allele`, `effect.allele`, `MAF`, `n.lines`, `nobs`, `n.missing`, `effect`, `SE`, `statistic`, `DF`, `p`, `score`, `p.BH`, `p.bonferroni`, `status` and `message`. Here `score` is minus log10 of the raw p-value and `DF` is the denominator DF (infinite for the Wald test).

When simulation is enabled, `results` additionally contains `p.sim`, and `simulation` contains:

`method, nsim, alpha, seed` The simulation method, requested replicate count, nominal level, and recorded master seed.

`thresholds` A data frame with columns `trait`, `alpha`, `nsim`, `successful`, `failed`, `n.markers`, `score`, `p`, and `status`. Status is `"ok"`, `"incomplete"`, or `"no.markers"`. `score` is the simulated cutoff and `p` its raw-p-value equivalent.

`max.score` A numeric matrix with `nsim` rows and one named column per trait. Failed replicates have missing maxima.

`diagnostics` A data frame identifying the trait, one-based replicate number, status, message, and marker if a marker test failed.

Without simulation, the ordinary result layout is retained and no `simulation` component is added. `as.data.frame` returns the marker results table; printing and plotting return the object invisibly.

compiled GWAS summaries

`compare.gwas.fits` summarizes successful finite marker tests separately for each model and trait. It returns the lead (smallest raw p-value) on each chromosome containing markers significant at `alpha` under the requested adjustment. This is a chromosome-level report, not a procedure for separating multiple linked peaks on one chromosome. Without discoveries it reports the overall lead; a trait with no successful tests has counts but no marker rows. `summarize.gwas.fit` returns the first `top` successful markers ordered by raw p-value, retaining additional significant chromosome leads. Ties retain original result-row order. No fit is refitted. `summarize.gwas.simulation` uses `p.sim` and rejects absent or incomplete calibration for any selected trait. It requires status ok, all requested simulations successful, zero failures and valid adjusted probabilities. All counting, ordering, selection and validation use compiled C workspace.

References

Endelman JB (2011). Ridge regression and other kernels for genomic selection with R package rrBLUP. The Plant Genome 4:250–255.

Yu J et al. (2006). A unified mixed-model method for association mapping that accounts for multiple levels of relatedness. Nature Genetics 38:203–208.

Kang HM et al. (2010). Variance component model to account for sample structure in genome-wide association studies. Nature Genetics 42:348–354.

Joo JWJ, Hormozdiari F, Han B, Eskin E (2016). Multiple testing correction in linear mixed models. Genome Biology 17:62. doi:10.1186/s1305901609036

Phipson B, Smyth GK (2010). Permutation P-values should never be zero: calculating exact P-values when permutations are randomly drawn. Statistical Applications in Genetics and Molecular Biology 9:Article 39. doi:10.2202/15446115.1585

---

st.hybrid.data.set *supported Hybrid Data Sets*

Description

Stores hybrid performance data with compiled references to two existing parental marker data sets, for symbolic GCA and SCA mixed models.

Usage

```
st.hybrid.data.set(markerdata1, markerdata2, hybrid.performance,
                   hybrid.data.set)
st.hybrid.data.set.list(digits = getOption("digits"))
## S3 method for class 'st.hybrid.data.set.list'
print(x, digits = NULL, ...)
st.hybrid.data.set.remove(hybrid.data.set)
st.hybrid.data.set.remove.all()
st.set.hybrid.performance.data(hybrid.performance, hybrid.data.set)
st.get.hybrid.performance.data(hybrid.data.set)
```

Arguments

`markerdata1, markerdata2`
: Names of two existing, distinctly named marker data sets. Their markers are referenced, not copied.

`hybrid.performance`
: A nonempty data frame with parental ID columns named exactly like the two marker data sets. IDs must identify existing parental individuals. Response columns, missing responses, repeated observations, and additional model variables are retained.

`hybrid.data.set`
: One nonempty hybrid-data-set name. Marker and hybrid data sets share a namespace. Reusing an existing hybrid name replaces its performance and parent references after validation.

`digits`
: Number of decimal places for printing. A printing value of `NULL` uses `st.options()$print.digits`.

`x`
: A hybrid-data-set listing.

`...`
: Additional printing controls, including `top`, `bottom` and `char.per.line`; omitted values use `st.options`.

Details

The performance frame is stored as a snapshot. Changing the original R frame does not change the supported data. `st.set.hybrid.performance.data` replaces the stored frame and `st.get.hybrid.performance.data` retrieves a snapshot.

`st.lmm` accepts an existing response column, for example `fixed=GY~1`, with a quoted stored name such as `data="hybrids"`. With parental datasets named `flint` and `dent`, `random=~flint+dent` fits GCA only. Adding `flint:dent` fits SCA. `ucov=list(flint="A", dent="A", "flint:dent"="kro(a,a)")` specifies genomic GCA plus SCA. The product is one quoted string; `"kro(A,A)"` is also accepted. Prepared covariance lists work identically. Unquoted matrix names refer to actual R objects. No R function called `kro` or caller matrix called `A` is needed for the quoted symbolic requests.

Each marker data set owns a dedicated `genomic_relationship_A` cache. A hybrid data set owns `genomic_relationship_SCA`, a factorized representation containing the parental references and unique-cross mapping. A selected SCA covariance is cached only when requested. The full Cartesian Kronecker matrix is never constructed. Repeated observations of a cross share its SCA effect.

Marker cache validity is checked against marker content, individual and marker identifiers and order, and coding. Marker changes invalidate related SCA caches. A response-only update retains the relationship caches. Changing unique parental pairs invalidates the selected SCA cache. Compacting a parental data set clears disposable caches; they are reconstructed on demand. A referenced parental data set cannot be removed until its hybrid data sets have been removed. Removing a hybrid data set does not remove its parents. Existing fitted models retain their own snapshots.

Listings report parental names, observation and unique-cross counts, both parental cache flags, factorized and selected SCA cache flags, and estimated payload memory in MiB. Shared parental caches are counted only in their marker data sets. Factorized mappings and stored performance belong to hybrid `ram.core`; selected numeric SCA caches belong to hybrid `ram.work`.

The accessor `st.lmm.components(fit, "predictions")` returns the stored hybrid prediction table with both parental identifiers. For `prediction = "NAonly"`, `source.row` indexes the retained performance data and identifies rows with missing responses. In a multivariate fit, observed trait cells are left missing in this prediction table. For `prediction = "Full"`, the table contains all combinations of the two parental pools, with the second parent varying fastest and `source.row = NA`. Full prediction requires non-parental fixed and random covariates to be constant. The call `predict(fit, newdata = ..., type = "conditional")` specifies other covariate scenarios. Unrecorded SCA effects are predicted directly from the parental factors and saved model scores, without inverting a genomic kernel. The default mode is `"NAonly"`.

Value

The constructor invisibly returns the supported name. The update function invisibly returns that name. The getter returns the stored performance frame as a copy. Listing returns a data frame of class `st.hybrid.data.set.list`. Removal functions invisibly return the number of hybrid data sets removed.

See Also

`st.lmm`, `st.get.A`, `st.data.set.list`, `st.compact.data.set`

`st.id` *Constructs a path in the SelectionTools input directory*

Description

Constructs 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` *Constructs a path in the SelectionTools input directory*

Description

Constructs 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 *Prints or handles a SelectionTools information message*

Description

Prints or handles 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 compiled 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 *Keeps 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 retain only 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 subset 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.kernel.structure` *Verify Kronecker structure in a relationship kernel*

Description

Detects a complete, ordered Kronecker product, or verifies a product of two supplied parental relationship matrices on an explicit set of crosses. Every matrix entry is checked. Detection does not replace a fitted kernel.

Usage

```
st.kernel.structure(K, factors = NULL, index = NULL,
                    dimensions = NULL, tolerance = 1e-10)
```

Arguments

`K`	A finite, symmetric square numeric relationship matrix with nonnegative diagonal. Row and column names, when supplied, must identify the same unique levels; columns are aligned to row order. A failed structure search does not certify positive semidefiniteness.
`factors`	Optional list of exactly two positive semidefinite parental relationship matrices. Their row and column names identify parental levels; unnamed matrices receive positional labels `1,2,...`.
`index`	When `factors` is supplied, a matrix or data frame with one row per level of `K` and two columns in factor-list order. Entries are parental names or one-based integer indices. Rows follow the row order of `K`. Missing, out-of-range and unknown levels are rejected.

`dimensions` Optional pair of integer factor orders greater than one whose product is the order of `K`. Without it, all nontrivial ordered splits are tried until a verified factorization is found. Cannot be combined with `factors` and `index`.

`tolerance` Maximum absolute reconstruction error divided by the maximum absolute entry of `K`; between zero and `1e-6`. Default `1e-10` allows numerical rounding. Approximate low-rank fitting is not performed.

Details

For a complete matrix in the corresponding order, $K = A \otimes B$ has blocks proportional to B. The function recovers candidate factors from principal submatrices and verifies the full reconstruction without constructing a second full Kronecker matrix. Unknown arbitrary permutations are not searched.

With supplied factors and indices it checks $K_{ij} = cA_{d_i,d_j}B_{f_i,f_j}$ for a positive scale c. This also describes incomplete crosses and replicated crosses. A subset of a Kronecker product need not itself admit a complete ordered factorization. Zero kernels return no identified product because their factorization is unidentified. Factors are not unique in general, including their reciprocal scaling; the function reports one verified representation.

`st.lmm` automatically checks pairs of supplied relationship terms against other relationship kernels, using compiled grouping indices. It requires a consistent parental assignment for every target-kernel level. Rows excluded for missing predictors do not establish mappings; missing phenotypes may do so. Unmapped target levels are not guessed. It then tries complete ordered products for unmatched kernels. The automatic mapped search checks at most 256 candidate pairs per model; attributes `candidates.checked` and `search.truncated` describe that search. The explicit function has no candidate-pair limit.

Automatic results are available as `fit$model$kernel.products`, `fit$model$specification$kernel.products` and `fit$engine$diagnostics$kernel.products`. The controls `detect.kernel.products=FALSE` and `kernel.product.tol` disable or set the tolerance of automatic detection. The metadata describes relationship matrices, independently of estimated variance components or coefficient bases. It is retained on serialization. In eligible scalar models, `st.lmm` uses this result for a complete-crossing tensor solver or an observed-cell spectral base with a smaller correction for other random terms. The selected route retains the supplied covariance; it does not substitute a reconstructed product. `kernel.solver=FALSE` keeps detection while disabling these computational routes. Further details appear in `st.lmm` for eligibility, numerical verification and diagnostic details.

Value

A list with `type` (`"none"`, `"kronecker"`, or `"mapped.kronecker"`), `factors`, `index`, positive `scale`, `factor.dimensions`, `complete`, `max.absolute.error`, `relative.error`, `tolerance`, `checked.entries` and `reason`. Indices refer to rows of the returned factors. `complete` requires every parental pair exactly once. No-match results contain no factor or index guess. Automatic detection returns a named list containing the verified matches only.

See Also

`st.lmm`

Examples

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
relationship <- st.get.A("maize.example")
A <- relationship[1:2, 1:2]
B <- relationship[3:5, 3:5]
z <- st.kernel.structure(kronecker(A, B), dimensions = c(2, 3))
z$type
z$relative.error
```

---

st.LDheatmap *Plots 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. TRUE draws a line representing the supplied marker positions and connects each marker in the equally spaced LD matrix to its position on that line.
add.key	Logical. TRUE draws 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. TRUE prints 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

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, ]
st.load.vcf.data(st.dataframe.to.STvcf(marker.frame), data.set = "maize.ld")
Z <- gs.get.Z("maize.ld", names = TRUE)
LD <- cor(Z, use = "pairwise.complete.obs")^2
map <- st.get.map(data.set = "maize.ld")
st.LDheatmap(LD, map$Pos, title = "Tropical maize LD",
             col = grDevices::heat.colors(20))
st.LDheatmap(LD, map$Pos, depth = 2, write.snp.id = TRUE)
```

st.LDplot.ld *Linkage-disequilibrium information for plotting*

Description

Prepares 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 *Linkage-disequilibrium information for plotting*

Description

Prepares 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.lmm *Gaussian Linear Mixed Models*

Description

Fits univariate and multivariate Gaussian mixed models with formula-based specification, covariance estimation and inference.

Usage

```
st.options(options = NULL)
st.lmm(fixed, random = NULL, cov = ~units, data,
       method = "REML", ddf = "Satterthwaite", type = 3L,
       random.test = "auto", control = list(), ucov = NULL,
       prediction = "NAonly", selection = NULL, weights = NULL,
       contrasts = NULL, subset = NULL, na.action = NULL, offset = NULL,
       rcov = NULL, series = NULL, constraints = NULL)
## S3 method for class 'st.lmm'
print(x, digits = NULL, top = NULL, bottom = NULL,
  char.per.line = NULL, ...)
## S3 method for class 'st.lmm.result'
print(x, digits = NULL, top = NULL, bottom = NULL,
  char.per.line = NULL, ...)
## S3 method for class 'st.lmm'
summary(object, selection = TRUE, ...)
## S3 method for class 'st.lmm.selection.summary'
print(x, digits = NULL, top = NULL,
  bottom = NULL, char.per.line = NULL, ...)
## S3 method for class 'st.lmm'
plot(x, scale = NULL, labels = NULL, main = NULL,
  col = NULL, which = NULL, type = "conditional", id.n = 3L, ...)
## S3 method for class 'st.lmm.selection.summary'
plot(x, scale = NULL, labels = NULL,
  main = NULL, col = NULL, ...)
## S3 method for class 'st.lmm'
anova(object, effects = "fixed", ...,
                       refit = TRUE, test = "Chisq", model.names = NULL)
## S3 method for class 'st.lmm'
coef(object, id = NULL, vcov = FALSE, ...)
## S3 method for class 'st.lmm'
vcov(object, adjusted = NULL, id = NULL, ...)
## S3 method for class 'st.lmm'
fitted(object, ..., type = "response")
## S3 method for class 'st.lmm'
residuals(object, type = "response", ...)
## S3 method for class 'st.lmm'
predict(object, newdata = NULL, type = "marginal",
```

```
                        se.fit = FALSE, allow.new.levels = FALSE,
                        new.levels = NULL,
                        id = NULL, offset = NULL, re.form = NULL,
                        newparams = NULL, ...)
## S3 method for class 'st.lmm'
logLik(object, ...)
## S3 method for class 'st.lmm'
AIC(object, ..., k = 2)
## S3 method for class 'st.lmm'
BIC(object, ...)
## S3 method for class 'st.lmm'
confint(object, parm, level = .95, method = "Wald", ...)
## S3 method for class 'st.lmm'
nobs(object, ...)
## S3 method for class 'st.lmm'
df.residual(object, ...)
## S3 method for class 'st.lmm'
formula(x, id = NULL, ...)
## S3 method for class 'st.lmm'
model.matrix(object, ...)
```

Arguments

options	A named list of shared printing options, or NULL for the current values. All supplied entries are validated before any value changes.
constraints	NULL, or a named list with fixed and/or shared components imposing physical covariance-parameter constraints. fixed is a named numeric vector; shared is a list of character vectors of parameter selectors. Further details appear in 'Fixed and shared covariance parameters' for the selector inventory, units, supported structures and inference limits.
series	NULL, or list("env", fit1, fit2, ...) containing at least two independent earlier ML/REML fits. The first string names the series factor; the remaining elements are ordinary fitted st.lmm objects with the same simple response name, or the same set of distinct response names in cbind(y1,y2,...). Trait order may differ between earlier fits. Exactly one fixed factor shared with the next-stage model identifies the adjusted means. Additional predictors in the next-stage fixed/random formulas are retained when they are nonmissing and constant within every trial. Further details appear in 'Series of earlier fits'.
rcov	NULL, or a known residual covariance matrix on the absolute response scale. Supported for univariate and multivariate ML and REML fits without precision weights. cov=NULL fits exactly this residual covariance; the default cov=~units adds an estimated independent residual variance. Further details appear in 'Known residual covariance'.
fixed	Two-sided fixed-effects formula. A matrix response such as cbind(y1,y2) creates a separate fixed-effect coefficient vector for each trait.
random	One-sided grouping formula, a random-regression formula, or a list of these. For example, ~gen + env + gen:env, ~Time | Chick, ~diag(1+Time) | Chick,

or ~1+Time || Chick. The double-bar form specifies independent expanded random coefficients and also works in additive bar terms and formula lists. Univariate random-coefficient models also support homdiag(), cs() and ar1(); further details appear in 'Random-coefficient covariance structures'.

cov — One-sided residual covariance formula, or a list of such formulas; default ~units. Terms containing units describe independent data rows, with possible residual covariance between traits on the same row. For a multivariate response, ~units estimates an unstructured trait covariance matrix. ~diag(trait):units specifies separate trait variances with zero residual covariances, ~at(loc):units specifies an unstructured trait covariance matrix per location, and ~diag(trait):at(loc):units specifies a separate variance per trait and location. Structured trait residuals also support homdiag(trait), cs(trait, hom=TRUE) and ar1(trait), combined with units. For multivariate repeated records, ~us(trait):ar1(Visit) | Subject combines one unstructured trait matrix with AR1 occasion correlation; the occasion factor also supports homogeneous CS and homdiag. Further details appear in 'Multivariate repeated records'. For repeated observations of a univariate response, a full factor indicator basis and an independent group are required, for example ~ar1(0 + Visit) | Subject. This form replaces units and allows residual covariance between different data rows in the same group. The declared levels of the factor Visit determine the occasion order. The supported structures and restrictions are described in 'Covariance formula syntax'.

data — Named data frame or list containing the model variables, or the name of a SelectionTools marker data set. A stored marker or hybrid data set must be supplied by its quoted name, such as data = "maize". Unquoted data = maize evaluates the R object maize; an undefined name is an error. Marker datasets supply y (stored performance values) and gen (individual identifiers).

method — "REML" (default), "ML", or "moments". The method of moments uses least squares for fixed-effects models and unconstrained moment variance and covariance estimates for supported balanced mixed designs, including multivariate responses. With subset comparison, "REML" still fits the specified full model by REML, but candidate fixed-effect subsets are compared using ML. This distinction is reported in the output; further details appear in 'Fixed-effect subset comparison'. For confint, method="Wald" retains the default fixed-coefficient interval; method="profile" constructs ML likelihood-profile intervals for selected physical parameters; further details appear in st.lmm.profile.

ddf — "Satterthwaite" (default), "Kenward-Roger", or "asymptotic" for likelihood fits. Kenward–Roger requires REML and a supported linear covariance structure; further details appear in 'Random-coefficient covariance structures'. With "Satterthwaite", fixed-factor ANOVA terms in complete balanced orthogonal categorical designs use an exact expected-mean-square ratio where available and are labelled "exact F". Other terms retain Satterthwaite inference. Explicit "Kenward-Roger" and "asymptotic" choices retain their requested reference. This automatic ANOVA choice does not change least-squares means or arbitrary coefficient-hypothesis inference. For method="moments", "Satterthwaite" is the default: single-stratum linear functions retain exact DF, and functions combining strata use Satterthwaite DF. "exact" requires exact inference. The model's design-specific ANOVA F tests retain their exact denominators.

type — Fixed-effect test type: 1, 2 or 3 (default). In predict, "marginal" uses fixed effects and offsets, "conditional" adds selected fitted random effects, and "random" returns only selected random effects. Diagnostic plots support "conditional" and "marginal". For fitted and residuals, "response" uses the fitted value or residual for the method's default target. This target is the conditional mean for likelihood mixed models, the ANOVA projection for mixed moments fits, and the marginal mean for models without random effects. "marginal" selects the fixed mean, "conditional" selects the mean including predicted random effects when available, and "projection" selects the ANOVA projection for moments fits. Likelihood fits also provide residuals(type = "pearson"). Further details appear in st.lmm.diagnostics.

random.test — "auto" (default) uses exact F-tests when a complete balanced orthogonal categorical design provides one denominator mean square for every random term, with an intercept, independent random intercepts and homogeneous independent errors. These automatic exact tests apply to unweighted models without offsets or supplied covariance matrices. Otherwise it uses "LRT". The explicit choice "F" requires those exact tests; an unavailable exact denominator is an error. These tests use the same balanced projection and expected-mean-square calculations as the moments path, without changing the ML/REML fit or fitting a moments model. The exact-test table reports DF1, DF2, F and p, with method="exact F". "LRT" explicitly tests each whole random term using an approximate chi-squared likelihood-ratio test, with one reduced-model fit per term. "bootstrap" requests a parametric bootstrap under the model without that term; "none" skips random-effect tests and their printed section. For method="moments", the default is "F" for univariate responses and "none" for multivariate responses. The supported choices are "F" and "none". Multivariate "F" requests separate tests by trait. An available single denominator mean square gives an exact F-test. The general path can also use quasi-F tests based on mean-square combinations when control$quasi.f is enabled.

control — Named list of fitting controls: optimizer="auto", ai.maxit (default 50), profile.scale=TRUE, maxit (default 500), factr, pgtol, lmm, trace, REPORT; bootstrap sets the replicate count (default 999). For a multivariate mixed model fitted by the method of moments, print.mean.products = TRUE includes the observed mean-product matrices (SSCP divided by DF) in the printed output. The default is FALSE. For an earlier-fit series, series.id="plot.id" names a common observation-ID column and checks that observations contributing to the first-stage fits have unique, disjoint IDs. These must identify observations globally, not merely enumerate rows separately in each trial. Further details appear in 'Series of earlier fits'. The compiled convenience policies are scale="none", singular="current", singular.tol=1e-4, boundary="current", and start="auto". Scale choices are "none", "check", and "auto"; all retain the engine's internal numerical scaling, while "check" warns for a scale ratio above 1000. Singularity choices are "current", "ignore", "warning", and "error". Boundary choices are "current", "accept", "restart", and "error". Start choices are "auto", "default", "zero.correlation", and "moments". The automatic and moment policies use a projected moment initializer for eligible univariate GCA-only models with two crossed scalar relationship terms, one scalar independent residual, constant coefficient loadings, at least 12000 observed re-

sponses, and backend="auto" or backend="sparse". Automatic selection also requires the combined relationship border to be smaller than the observed sample. Other combinations retain the existing feasible initializer, including an explicit start="moments" request. Supplied covariance starts and parameter evaluations retain their values. Diagnostics start.moments.status and start.moments.rows describe the path used. This initialization policy is separate from method="moments". joint=TRUE additionally stores joint fixed/random prediction-error covariance, including with ddf="asymptotic". backend="auto" retains complete spectral product shortcuts. For eligible incomplete scalar GCA/SCA products with 6000–12000 observed responses and at least 16 configured threads, it prefers the dense calculation. Other models retain the spectral, sparse mixed-model, covariance-block and dense eligibility rules. The thread count supplied by st.set.num.threads is preserved. backend="dense" retains that reference calculation. backend="spectral" requires an eligible spectral product model. eigen.solver="auto" prefers DSYEVD within its workspace limit; "dsyevd" and "dsyevr" select the driver explicitly. backend="sparse" or backend="block" forces the respective route without its size heuristic; unsupported structures are rejected. A singular base covariance falls back to the dense calculation.

ucov — NULL, or a named list of known covariance matrices for random grouping terms, for example list(gen = A) for a genomic relationship matrix. With a stored marker dataset, the quoted string "A" requests its internally constructed VanRaden matrix; a stored hybrid interaction uses "kro(a,a)". Unquoted A evaluates the actual R matrix. Prepared lists are supported and described in the Details section. Each entry must be a finite, symmetric positive-semidefinite square numeric matrix with unique nonempty row and column names identifying the same levels. Matrices are matched by names, including when column order differs from row order. Every non-omitted grouping level must be covered. Extra matrix levels are retained for prediction. Unmatched or duplicate list names are errors. Without an entry, the corresponding grouping levels remain independent.

x, object — An st.lmm object, an st.lmm.selection.summary object for its print and plot methods, or an extracted result for the st.lmm.result print method.

parm — Coefficient names or indices for confidence intervals; omitted for all coefficients.

level — Confidence level for fixed-coefficient intervals.

model.names — Optional unique nonempty names for compared models. NULL uses the supplied model expressions where concise and distinct, with numbered labels as a fallback.

refit — For multi-model anova, TRUE refits REML models by ML. Further details appear in st.lmm.compare.

test — For effects = "manova", an omitted argument or NULL returns all stored methods. The available method names are "Pillai", "Wilks", "Hotelling-Lawley" and "Roy". For multi-model anova, "Chisq" (default) requests likelihood ratio inference where nesting is established. For fixed-effect models fitted by the method of moments, the default and "F" request exact F tests. The choice "none" omits tests in either comparison.

`k` Penalty per parameter in AIC; default 2.

`prediction` For a SelectionTools hybrid data set, `"NAonly"` (default) predicts missing responses in the retained performance frame and `"Full"` predicts every combination of the two parental pools. The stored table and parental identifiers are available through `st.lmm.components(object, "predictions")`. Further details appear in `st.hybrid.data.set` for prediction coverage and covariate requirements. This option does not change non-hybrid fits.

`selection` For `st.lmm`, NULL leaves subset comparison disabled; a named list enables fixed-effect subset comparison. The list controls are described in 'Fixed-effect subset comparison'. For `summary`, a logical value: TRUE (default) returns the subset comparison summary when the fitted object contains a search, while FALSE returns the ordinary summary of the specified full model.

`weights` NULL or a numeric vector of observation precisions, evaluated in `data` and then the calling environment. With homogeneous independent residuals, positive weight w_i gives residual variance σ^2/w_i. With random effects or structured residuals, weights require a univariate response and `"ML"` or `"REML"`; every retained row, including a missing-response prediction row, must have a finite, strictly positive weight. Fixed-effects-only regression with `cov=~units` also supports `"moments"` and preserves its existing exclusion of zero and non-finite weights. Finite negative weights are errors. These are precision weights, not observation replication counts.

`contrasts` NULL or a named list assigning contrast matrices or contrast functions to factor expressions. An entry may be a numeric matrix, a function, or a function name such as `"contr.treatment"`, `"contr.sum"`, `"contr.helmert"`, `"contr.poly"` or `"contr.SAS"`. It overrides the corresponding factor's contrasts attribute. Otherwise that attribute is used, followed by `getOption("contrasts")`; ordered factors use its second entry. Matrix rows correspond to the input factor levels in order. Matrix values must be finite; supplied column names must be nonempty. The fitted coding is retained for prediction and subsequent updates.

`subset` Optional row selection, evaluated in the input data and calling environment. A logical scalar or a logical vector with one value per input row is accepted. Integer indices may be positive, including repetitions, or negative; zeros are ignored and positive and negative indices cannot be mixed. Missing, fractional and out-of-range indices are errors.

`na.action` NULL preserves the package's missing-response prediction behavior, independently of the global `na.action` option. Alternatives are the standard functions `na.omit`, `na.exclude`, `na.fail` and `na.pass`, or their names as strings. Custom functions are not accepted. The explicit policies and row mapping are described below.

`offset` For fitting, an optional known contribution to a univariate response mean, evaluated in the input data and calling environment. It is a numeric vector with one value per input row, or a scalar repeated over rows. It is added to any fixed-formula offsets. For prediction, an explicitly supplied value replaces only this argument-offset component; an explicit NULL removes it. Prediction offsets have one value per new-data row, or per retained model-frame row when `newdata=NULL`, or may be scalar. Omitting the prediction argument uses the fitted offset recipe when possible.

`id` NULL (default) refers to the specified full model. A subset model identifier selects a retained candidate for coefficients, covariance, formulas, predictions or Wald intervals. Identifiers correspond to rows of the subset summary. A single identifier or a vector of identifiers returns a list named by identifier, in the requested order. For `confint`, `id` is supplied through `...` with `method = "Wald"`.

`vcov` For candidate extraction with `coef` and `id`, attach the coefficient covariance matrix when TRUE. The default is FALSE.

`scale` Criterion used to order the subset inclusion plot: `"rss"`, `"r2"`, `"adjr2"`, `"Cp"`, `"BICm"`, `"AIC"`, `"BIC"` or `"logLik"`. NULL uses `"BICm"` when available and `"BIC"` otherwise.

`labels` Optional labels for the columns of the subset inclusion plot, one per inclusion column, including the intercept when fitted. For a diagnostic plot, one character label per retained response cell, in response-matrix column order. Missing labels are not annotated.

`main` Optional plot title. For several diagnostic panels, either one title or one title per panel.

`col` Optional colour vector for the subset inclusion plot. For diagnostic plots, one colour or one per retained response cell.

`which` NULL retains the subset inclusion plot when the model contains subset results and otherwise requests four diagnostic panels. `"selection"` explicitly requests the subset plot. Diagnostic selectors are `"residuals"`, `"qq"`, `"scale-location"` and `"leverage"`, or their indices 1 through 4. Further details appear in `st.lmm.diagnostics` for their definitions.

`id.n` Maximum number of diagnostic observations to annotate, chosen by absolute standardized residual. Zero suppresses annotations. This argument does not change subset inclusion plots.

`adjusted` For `vcov`, NULL uses adjusted covariance when the fit requested Kenward–Roger. TRUE requests that adjustment explicitly; FALSE returns the model-based covariance.

`digits` Number of decimal places for printed statistics and p-values (integer 1–16). `NULL` uses `st.options()$print.digits`, initially 4. Degrees of freedom use at most two decimals and print as integers when they round to an integer at that precision. Counts use integers. Positive values below the display precision are shown as an upper bound, for example `<0.0001`. Displayed F-statistics above 1000 are shown as `>1000`. Stored numeric values remain unchanged.

`top, bottom` Nonnegative integer numbers of leading and trailing rows to display in each table. `NULL` uses `st.options()$print.top` and `st.options()$print.bottom`, initially 10 and 5. Tables with no more than their sum are shown in full. Otherwise an explicit omitted-row count separates the two parts. Either argument may be `Inf` to show every row. Zero suppresses that part. The model summary uses separate settings for the number of covariance blocks.

`char.per.line` Maximum output line width (integer 40–10000). `NULL` uses `st.options()$print.char.perline`, initially 75. The printer first reduces spacing, then decimal places (to at least two, or one if explicitly requested), then abbreviates labels with a printed key.

Probabilities and existing numerical bounds keep the requested precision. If needed, it splits wide tables into column blocks with repeated row numbers and the first column. Extremely wide numbers use wrapped row records. Prose and formulas also wrap. UTF-8 widths are measured conservatively in bytes without splitting a character. Printing does not change stored numbers or labels.

`effects` `"fixed"` and `"random"` return the corresponding cached test table. For `"random"`, a model without random terms returns a zero-row table. Random terms can appear without a computed test. Both `"both"` and `"all"` return one data frame of stored fixed, random and MANOVA tests with the same columns for every fit method. Its `scope` distinguishes individual traits, joint trait hypotheses and whole terms. The reference distribution, degrees of freedom, test statistic, `p` and denominator source are recorded where available. The `statistic` and `calculation` columns retain the original MANOVA measure and its F reference. The value `"manova"` returns only the stored MANOVA table for one fitted model. Its columns are `term`, `method`, `statistic`, `DF1`, `DF2`, `F`, `p` and `calculation`. The print method omits intercept tests by default and identifies the term in the heading when a single term is displayed. A fit without stored MANOVA tests gives an error. A second fitted `st.lmm` object starts a multi-model comparison.

`newdata` Optional data frame for prediction. Fixed-factor levels and retained transformations follow the fitted model. Only the predictors and grouping variables used by the requested prediction are required. A random-only prediction does not require fixed-effect predictors or offsets.

`se.fit` The prediction list always contains a `fit` matrix. TRUE adds a `se.fit` matrix and, for ML and REML, a logical `estimable` matrix. All three matrices have one row per observation and one column per trait, including fits with one trait. For moments fits with standard errors, the list also contains `DF` and `residual.scale`. A requested moments interval appears separately in an `interval` matrix with columns `lwr` and `upr`. The common printer displays univariate diallel overlay predictions together in one table. Diallel overlay results include readable cross labels such as P2:P4; these are display metadata, and the returned numerical components are unchanged. For ML and REML fits, standard errors describe the prediction error of the requested latent mean, including fixed-effect estimation when those coefficients are estimated. They exclude future residual noise and covariance-parameter uncertainty. Further prediction details appear below.

`allow.new.levels`
TRUE allows previously unseen independent random-effect levels with a predicted random contribution of zero. Their fitted prior variance remains in conditional and random-only prediction uncertainty. For a previously unseen covariance stratum, only point prediction with a zero random contribution is available. Levels absent from a supplied known relationship matrix remain errors; known parental relationships can define new hybrid combinations.

`new.levels` Optional string convenience policy for unseen grouping levels. `"error"` is equivalent to `allow.new.levels=FALSE`; `"population"` is equivalent to `allow.new.levels=TRUE` and gives an unseen level zero random contribution. `NULL` preserves the existing `allow.new.levels` argument and its default.

re.form — For ML and REML conditional or random-only predictions, NULL includes all fitted random coefficients. NA, ~0, ~-1 or character(0) includes none. A character vector selects complete covariance blocks by their exact names in st.lmm.components(object,"random"). A one-sided random formula selects exact fitted coefficients, for example ~1|group or ~0+x|group from a fitted correlated intercept and slope. Parenthesized sums such as ~(1|group)+(1|block) are also accepted. Selected coefficients must have exactly the fitted basis and coding. The default marginal prediction excludes all random effects; a nonempty explicit selector requires type="conditional" or type="random".

newparams — Optional physical parameters for ML or REML prediction: a named list with fixed and/or covariance. The latter is a named list with random and/or residual, each naming every covariance block in that section. Fixed coefficients must be a finite named vector containing every fitted model-matrix column, including aliased columns. Covariances must be named symmetric positive-semidefinite matrices in the original coefficient units and must obey the fitted covariance family. Omitted sections retain fitted values. Names, dimensions and family constraints are checked, and coordinates are aligned by name. Further details below describe the distinction between supplied and estimated fixed coefficients.

... — Additional method arguments. For univariate fixed-effects fits using method="moments", anova accepts further nested fits. For univariate ML or REML fits, anova accepts further fitted models and drop1 performs fixed-term deletion; further details appear in st.lmm.compare. Fixed-effect subset comparisons use the selection argument of st.lmm. Additional subset-summary arguments are described below.

Details

Half diallel and parental overlay: random=~overlay(mother, father) creates one shared parental effect vector. Each cross receives the sum of its two parental effects; a self cross has loading two. The parental union is shared across both roles, with one estimated GCA variance. An optional parent relationship matrix is specified as ucov=list(overlay=A) or under the exact overlay term name. Additional parent levels in this matrix are retained for prediction. st.diallel.cross constructs an unordered cross factor that can enter as a separate random term for SCA. Reciprocal and maternal effects require separate specification. The printed variance-component table reports the shared overlay variance once, rather than expanding it into a repeated diagonal matrix. The stored covariance matrix and fitted model are unchanged.

Univariate overlays can be standalone, interact with ordinary directly named grouping factors, or be stratified by at(). The parental union is common to both roles and all grouping levels or strata. A supplied parental relationship matrix applies to every group and stratum; unused parent levels in that matrix remain available for prediction. Parent roles may have different factor level orders, and reversing the roles leaves the model unchanged. A self cross retains loading two in every active stratum.

Random formula	Covariance interpretation
~overlay(mother,father)	One shared parental-effect vector.
~overlay(mother,father):env	Independent parental-effect vectors in each environment, with one comm
~at(env):overlay(mother,father)	Independent environment strata, each with its own parental variance.
~at(env,"E2"):overlay(mother,father):year	Parental effects only in E2, independent across years, with one variance

`~overlay(mother,father):env:year` Independent parental effects for environment-by-year groups, with one c

Writing h_i for the two-parent incidence vector of observation i, a grouping interaction has covariance $\sigma^2 h_i^T A h_j I(g_i = g_j)$. Its coefficient covariance is the product of an independent grouping identity and the parental covariance. An `at()` term adds the stratum indicators and estimates a separate σ_s^2 for each selected stratum. Multiple `at()` factors form their selected Cartesian strata. Ambiguous combined stratum labels are rejected and require distinct level labels before fitting. No correlation across grouping levels is inferred. A common interaction variance alongside all its separate stratum variances defines redundant covariance directions and is rejected. Covariance-direction checks include fixed-effect projection for REML. Active fixed/shared constraints use the existing reduced checks.

The common parental relationship is specified by `ucov=list(overlay=A)`, the exact overlay expression name, or its complete interaction term name. These keys identify parental covariance; they do not specify covariance between environments or years. A different matrix for each selected stratum is not inferred. The representation stores a dense observation-by-parent basis and parental covariance, so memory grows with observations and parent count.

`st.predict` with the complete fitted overlay block name returns intercept plus parental GCA and its full joint prediction-error covariance; `"0 + "` requests GCA alone. Grouped results list parents within each independent group. Factor-level labels use byte-length prefixes to avoid ambiguous concatenation, followed by the parent label. Stratum-specific blocks are selected by their exact fitted names, available from `names(st.lmm.components(fit, "covariance")$random)`. Other fixed predictors are at their intercept reference; this is not an average of fixed environment effects. Conditional `predict` with cross data is the direct interface for complete cross means and their uncertainty.

Fitting, objective gradients, inference, refitting, updates, serialization, training/future cross prediction and simulation use the same layout. New independent grouping levels require `allow.new.levels=TRUE`; their parental effects have zero conditional mean and the fitted prior covariance. New parents must already be represented by the retained parental union or known relationship matrix. New `at()` strata do not acquire new variance parameters automatically.

Multivariate overlays, residual overlays, multiple distinct overlays in one interaction, trait modifiers and units within overlay interactions, random coefficient formulas and correlated coefficient products remain unsupported and fail explicitly. Separate additive overlay terms are allowed when their covariance directions are identifiable. No automatic maternal, reciprocal or cross-environment correlation is introduced.

Series of earlier fits: `series=list("env", fit1, fit2, fit3)` constructs the response data from earlier fitted models without refitting them. The fixed genotype factor is inferred as the single fixed predictor of the first fit shared with the new fixed/random formulas, excluding the series factor. It must be fixed and categorical in every earlier fit. Adjusted means and their full model-based covariance use the compiled least-squares means machinery, with equal nuisance-factor averaging and numeric covariates at its default reference values. Nonestimable means are rejected. Transformed responses, moment fits and model selection are outside this interface. Shared-predictor ambiguity can be resolved by an explicit genotype in the reference declaration.

A multivariate second stage has a response formula such as `cbind(y1,y2)~env+gen`. Each earlier fit must use those same untransformed response names, with a fixed genotype factor. Trait order is matched by name. All trait-specific genotype means and their full joint covariance are inherited; cross-trait and cross-genotype covariance within an earlier fit is retained. Covariance between

independent earlier trials is zero. Missing first-stage trait cells and incomplete genotype sets are allowed when every requested within-trial genotype-by-trait mean is estimable. Otherwise extraction fails explicitly. A row contributes to provenance and observation-ID checks once when any trait is observed. The automatic mean reference grid uses the rows contributing any observed trait, equal nuisance-factor averaging, and the usual numeric reference values. Explicit reference grids and weights can replace that choice, as described below. The reference definitions determine the targets when covariate distributions differ.

The second-stage data have one row per earlier-trial genotype mean and one response column per trait. Known covariance is in trait-major order, with all data rows for trait one followed by all rows for trait two. Its row and column names identify trait cells as described below. Subsetting preserves this order on both covariance axes. Refitting, updating, serialization, training-data simulation and multivariate `predict` retain this covariance. The existing univariate-only restriction in `st.predict` remains. Ordinary model inference conditions on estimated first-stage covariance parameters. For univariate independent trials, `st.lmm.bootstrap` with an explicit `series` declaration can refit both stages; its current scope and generation assumptions are documented there. Ordinary inference does not account for first-stage covariance-parameter estimation uncertainty. Cross-fit dependence must be absent or represented by the explicitly supplied joint covariance described below.

Additional trial-level predictors referenced in the new fixed or random formulas are copied from each earlier fit's retained data and repeated for its genotype means. Every predictor must be a nonmissing scalar column, constant within each trial, with consistent categorical (factor/character) or numeric interpretation across fits. Factors use their labels, so local level codes need not agree. Numeric columns retain their numeric meaning; Categorical years are represented by a factor. Predictors varying within a trial are rejected rather than silently averaged.

Known cross-fit target covariance. The default treats errors in distinct earlier fits as independent. `control$series.covariance` specifies the full known numeric covariance matrix of all inherited target cells on the absolute response scale. It is not a correlation matrix and has no estimated multiplier. Both axes must name every target cell exactly once; their orders may differ. The `cells` entry of `st.lmm.components(object, "series")` contains the complete cell declaration. Its columns are `label`, `trial`, `genotype`, `trait`, and `data.row`. Cells are in trait-major order, before any second-stage row subset. The labels encode the UTF-8 byte lengths and values as `length(trait):trait:length(trial):trial:length(genotype):genotype`. The returned labels identify cells without ambiguous separators. A default fit can expose the declaration; its independence-based inference is not valid evidence of independence when the caller knows dependence exists.

The supplied matrix must include all original target cells, even if the second-stage call selects fewer rows. Every within-fit block, including cross-trait and cross-genotype entries, must agree with the extracted covariance to absolute tolerance 10^{-8} on the earlier-target correlation scale. Those blocks retain the exact extracted values. The full matrix must be finite, symmetric and positive semidefinite, checked after normalization by the earlier target standard deviations. Earlier target variances must be positive and finite. Observed marginal covariance still must be positive definite; no jitter or ridge is added. Tiny accepted cross-block asymmetries are averaged. Reference grids and weights define the targets before this alignment and validation.

This option handles externally known dependence, for example a shared calibration error across distinct observations. It does not infer or estimate cross-fit covariance from matching IDs or propagate uncertainty in estimated first-stage covariance components. Correct cross-fit values and meaningful target definitions remain the caller's responsibility. Copied fits, overlapping source provenance and duplicate observation IDs remain rejected: a supplied matrix does not authorize

reusing observations in separate earlier fits. The covariance.provenance entry of the series component records the supplied-covariance source, within-fit tolerance and pre-subset scope. The cell mapping and source are retained through refitting, updating, serialization and simulation. The rcov entry retains its fit-time dimnames, when present. The series.covariance control uses the complete set of labels in the cells entry.

Explicit target/reference grids. An explicit reference is specified by control=list(series.reference=list(genotype grid=grid,weights=weights)). The genotype name is optional when the usual single shared fixed-factor inference is unambiguous; declaring it resolves ambiguity when additional shared covariates occur in both stages. It must name a fixed categorical factor in every earlier fit and occur in the next-stage fixed or random formula. The multivariate virtual trait modifier cannot be the genotype target.

grid is either one common data frame for all trials, or a complete named list of data frames indexed by the resolved trial labels. Each grid contains the genotype column and every raw predictor used by the earlier fixed formula, excluding the multivariate virtual trait. No extra columns are accepted. Columns are scalar, nonmissing and finite where numeric; genotype targets use factor or character labels. All values are interpreted through the earlier fit's retained design schema, including learned transformations and factor contrasts. A common grid with x=20, for example, requests the same covariate reference in all trials rather than their different observed means. Trial-specific grids can accommodate trial-specific nuisance-factor labels or explicitly different references.

The requested genotype targets and their order are the distinct genotype labels in first-appearance order in each grid. The grid is used exactly as supplied; no Cartesian expansion or automatic nuisance-factor averaging is added. Replicate rows can express the desired joint distribution of nuisance factors and covariates. This permits valid nested targets that the automatic grid rejects. Unknown design levels and nonestimable weighted linear targets fail explicitly. Individual rows need not separately be estimable when their weighted linear combination is estimable; the caller must declare meaningful reference combinations.

weights=NULL gives equal weight to grid rows within each genotype. Otherwise one finite nonnegative numeric weight is required per row. A named vector is aligned independently to the grid's row names; names must match exactly once. A complete named list of numeric vectors or NULL values can supply trial-specific weights. Weights are normalized within genotype, not across genotype means or trials; every target needs positive total weight. Uniform rescaling within a target leaves the result unchanged. Zero-weight rows still undergo grid and design validation. The full joint covariance is computed from the same weighted linear functions as the means. These are reference-distribution weights, not residual precision weights. A trial's grid and weights apply to all response traits; trait-specific reference distributions are not part of this interface.

For example, expand.grid(gen=c("G3","G1"), block=c("B1","B2")) with a declared common x column and row weights describes explicit weighted nuisance-factor averages for the two targets. All fixed predictors required by each earlier model must be present. Per-trial grids need not request the same genotype set, but the second-stage model still needs sufficient connectivity and estimability.

When an additional next-stage predictor is present in the reference grid, its value must be constant over positive-weight rows within each genotype target. That target value is copied into the second-stage data. A varying nuisance factor cannot silently be represented by one category or averaged numeric value in a new formula. Other additional next-stage predictors retain the existing requirement of being constant within the earlier trial. The reference entry of the series component retains the declared grids and weights. Its genotype entry records the resolved genotype column. These declarations precede any second-stage row subset. Refits, updates and response simulations

retain them without re-extracting means or refitting the earlier models. With no explicit reference declaration, the automatic-grid behavior remains in use.

For repeated locations across years, each independent location-year trial has a separate fit, common loc and year identifiers, and a unique env trial label. A second-stage fixed formula y~env and random formula ~gen+gen:loc+gen:year+gen:env separates main genotype, persistent genotype-location, genotype-year and transient genotype-trial effects. Incomplete grids are allowed, but sufficient connectivity is needed to identify the requested effects. An independent genotype-trial variance can be redundant with an additional units variance, as checked below. The same covariance check applies to an equivalent gen:loc:year interaction. A supplied genomic covariance for gen does not automatically apply to its interaction terms. No temporal correlation or future-year forecasting is created merely by retaining year.

Each earlier fit must represent distinct observations from a trial. Errors are assumed independent across fits unless their known joint target covariance is supplied explicitly. No cross-fit covariances are inferred. Named fit entries supply environment labels when the environment column is absent. If that column is retained, it must have one nonmissing value on the observations used by the earlier fit, and a supplied entry name must match it. Named entries cannot combine multiple environments into one trial. Duplicate environment labels are rejected. The response, omission indicators, fixed-design dimensions and retained row map are checked before adjusted means are extracted. An unused matrix column does not determine the observation count.

New top-level fits carry an opaque provenance token which is preserved by serialization, `st.lmm.refit`, and `update` when the original data are retained. Their original row maps detect shared contributing observations. Copies and overlapping refits cannot be counted as independent trials; disjoint subsets of a common source can be supplied as separate trials. Separately fitted independent trials are not equated merely because their values happen to agree. Identical earlier snapshots without a provenance token are rejected when independence cannot be checked. Independence can be checked after refitting independent trials with the current version or through genuinely disjoint observation IDs as follows.

With `control=list(series.id="plot.id")`, every earlier fit must retain the scalar column `plot.id`. Its IDs must be nonmissing, nonempty and unique among the observations used for that fit. IDs shared by different fits are rejected, including after serialization. Factors are compared by labels; numeric IDs use a full-precision representation. Rows with missing responses or omitted model inputs do not contribute observations to this check. Relabelling shared observations does not establish independence; shared evidence requires a joint analysis.

The `independence` entry of the series component records the trial labels, observation counts, source provenance, ID column and whether ID disjointness was checked or independence remains the caller's assumption. Disjoint IDs do not prove that trial errors are independent: shared calibration and other cross-fit dependence require an externally known joint target covariance or a joint analysis. The response and its covariance are assembled in matching order, and existing subset and covariance-alignment handling applies thereafter. The data and rcov arguments cannot accompany series.

The printed residual description explicitly identifies the supplied covariance and absence of an additional units variance when cov is NULL. The inherited (or supplied) covariance is reported separately from estimated components, with its scale fixed at one; this is not a common error variance of one.

The inherited covariance is the block diagonal matrix of earlier adjusted mean covariances, with coefficient fixed at one. It conditions on the estimated earlier covariance components. Ordinary units covariance adds an estimated independent GxE variance. For scalar variance components, identifiability is checked using the actual covariance matrices on the retained observations.

REML checks project out the fixed-effect design. If the omitted default units component is the sole redundant covariance direction, it is omitted automatically. Explicit `cov=~units` is rejected when it is redundant. Identifiable alternatives include `cov=NULL`, omission of a redundant term, or fixed/shared constraints. A random environment main effect alone does not suppress units. A genomic GxE kernel can coexist with units when their projected covariance directions are independent. Raw matrix rank before accounting for fixed effects is not sufficient: a compound-symmetric GxE kernel can become proportional to units after fixed environment effects are removed. Fixed/shared constraints are checked in their independent feasible scalar covariance directions.

For the shared genetic random term, `st.predict` on a series fit adds the equally averaged fixed environment mean instead of the reference intercept. Its SE uses joint prediction-error covariance. The returned prediction has a `reliability` attribute, also shown in its summary, equal to genetic BLUP variance divided by prior genetic variance. This reliability excludes uncertainty in the added environment mean. Genotyped levels represented in the supplied relationship matrix are retained for prediction, including those without phenotypic means. Missing relationship matches remain errors under the ordinary ucov contract; no lines are silently discarded.

The maintained v-lmm companion contains the three-environment example. This implements the homogeneous GxE stagewise model of Endelman (2023), doi:10.1007/s00122-023-04298-x, using full earlier-stage covariance propagation as in Piepho et al. (2012), doi:10.1002/bimj.201100219. It does not implement StageWise's spatial splines or factor-analytic location covariance.

Known residual covariance: With `rcov=C1,cov=NULL`, the residual covariance is exactly `C1`, without estimating a multiplier. This propagates the complete covariance of first-stage adjusted means into a second-stage model. `as.matrix(vcov(means))` from `st.lsm` provides the covariance; its row names identify the corresponding rows of the second-stage data. Diagonal precision weights discard these between-mean covariances.

The matrix must be finite, symmetric and positive semidefinite; the observed marginal model covariance must be positive definite. No automatic ridge or jitter is added. If either axis is named, both axes must have unique names matching `rownames(data)`. They are aligned independently before applying `subset` and missing-value exclusions to both axes. Without names the matrix is positional in the original data row order. For `cbind(y1,y2,...)` with n input rows and p traits, `rcov` is an $np \times np$ matrix on the original trait scales. Its positional order is all n rows for trait one, then all rows for trait two, and so on. Named axes use explicit trait-cell labels: `paste0(nchar(trait, type="bytes"), ":", trait, ":", rowname)`. The byte-length prefix makes labels unambiguous even when names contain colons. For example, trait `y1` on row `plot7` is `"2:y1:plot7"`. Both axes must contain the complete set exactly once; row and column permutations are aligned independently. Names use UTF-8 trait labels. Trait names must be nonempty, nonmissing and distinct. Missing responses select the observed principal submatrix without replacing unobserved responses. A row subset selects its entries within every trait, including every cross-trait covariance block. The covariance must include all original rows and traits before this selection.

An additional `cov` formula adds its estimated covariance components to the supplied matrix. The fit conditions on the supplied covariance; uncertainty in its first-stage estimation is not propagated.

The matrix is retained for updating, refitting and training-data simulation. Future residual simulation requires future covariance and is therefore rejected by the current newdata interface. Latent-mean newdata prediction remains supported. Known `rcov` is not supported by the method of moments or with precision weights.

Row selection and missing values: Response and formula expressions, including data-dependent

transformations, are evaluated on the original data before applying `subset` and the explicit missing-value policy. This follows `stats::model.frame`; a polynomial or spline is therefore calibrated on the full input even when only a subset is fitted. An expression that cannot be evaluated on the full input must first be supplied as a precomputed data column.

Explicit `na.omit` and `na.exclude` remove a common row whenever any referenced response, predictor, grouping variable, offset or precision weight is missing. For a matrix response, a missing trait removes that row for every trait. Unreferenced data columns do not affect this policy. `na.fail` rejects missing model inputs after subsetting. `na.pass` retains missing responses as prediction rows but requires finite complete predictors, offsets and weights. Nonmissing infinite responses are invalid. With an explicit policy, retained predictors and offsets must be finite. With the default NULL, missing responses and the historical predictor-omission behavior remain available.

For mixed models and structured residual models, every retained precision weight must be finite and strictly positive after the explicit NA policy. Ordinary weighted regression retains its existing exclusion of zero and non-finite weights; negative finite weights remain errors. Subset comparison additionally uses its common complete analysis sample.

Internal matrices and `model.frame` contain retained rows. With `na.exclude`, public `fitted`, `residuals` and predictions without `newdata` insert NA values at removed positions in the selected row order, including repeated selected rows. A new-data prediction uses only its new-data rows. The model retains original row IDs, unique selected row names, retained positions and separate NA, weight and common-sample omission maps in `model$input`. Further details appear in `st.lmm.model`.

Comparing fitted models: `anova(m1, m2, ...)` compares fitted univariate likelihood models. The default `refit=TRUE` converts REML fits to ML and reports this convention in the output. `drop1(m)` deletes eligible fixed formula terms while preserving hierarchy. Observation compatibility, nesting and variance-boundary qualifications are described in `st.lmm.compare`.

Offsets, contrasts and retained model information: An additive `offset(expression)` in a univariate fixed formula supplies a known contribution to the response mean. Multiple offsets are added; they do not create estimated coefficients or model-matrix columns. Estimation uses the offset-adjusted response, while fitted values and predictions are returned on the response scale. New-data predictions evaluate the retained offset recipe in the new data. Offsets cannot be random-coefficient terms or parts of formula interactions. Least-squares means include the known offset evaluated on their reference grid, and trends include its finite-difference contribution.

The explicit `offset` argument and formula offsets are stored separately and each contributes once. A scalar argument offset, or an expression based on data columns, can be reused for new-data predictions. Learned scaling and polynomial or spline recipes retain their training calibration. A training-only external offset vector cannot be safely matched to new data, even if the row counts happen to agree; an explicit prediction `offset` is required in that case. Least-squares means and trends also evaluate scalar or data-derived argument offsets on their reference grid. A training-only vector without a grid recipe is rejected by `st.lsm`; no arbitrary reference offset is assumed.

Factor contrast matrices are stored with their level mappings. Later changes to factor attributes or global contrast options do not change the fitted coding used for prediction. No-intercept terms use full indicator coding where required by the formula. Reduced or redundant user contrast matrices specify their actual model space; fixed-effect aliases are reported by the likelihood and fixed-only moments paths. For reduced or redundant coding, Type III construction centers the supplied columns over factor levels.

`model.frame`, `terms` and `weights` expose retained model information. `update` constructs a new fit using retained data and settings; `update(object, evaluate=FALSE)` returns its call. Further details appear in `st.lmm.model` for evaluation and persistence details.

Precision weights: For a supported structured residual covariance R_0, the supplied precisions define $R = W^{-1/2} R_0 W^{-1/2}$, where $W = \mathrm{diag}(w_i)$. Random-effect covariance is unchanged. The transformation also applies with known random-level covariance matrices, bitpacked SelectionTools marker data sets and SelectionTools hybrid data sets.

When all residual covariance parameters are estimated, multiplying every weight by the same positive constant rescales those parameters by that constant and leaves the fitted observation covariance, coefficients, predictions and likelihood unchanged. The reported residual variance therefore refers to a weight of one. Missing-response prediction rows in mixed models remain in the data and require valid positive precisions.

Fixed-effect subset comparison: Supplying `selection` compares subsets of the fixed regression part of a univariate model. The selection unit is explicitly one original column of `data`, not one column of the expanded model matrix. All contrast columns of a categorical factor enter or leave together. Transformations using the same source column also enter or leave together; for example, `x` and `I(x^2)` form one selection unit when both occur in the full formula. An interaction or other term involving several source columns is retained only when all of those source columns are selected. Selection does not add terms absent from the supplied full formula. This grouped search deliberately differs from procedures that select individual factor contrasts. The full fixed formula must reference at least one candidate data column. At most 63 original source columns are supported by the subset search.

The fitted object continues to represent the full model specified by `fixed`, `random` and `cov`. Subset results are attached to `object$selection`; no candidate automatically replaces the fitted model. The default `summary` shows the comparison when a search is present. `summary(object, selection=FALSE)` provides the ordinary full-model summary. The user may inspect the comparisons and then make another `st.lmm` call with the chosen formula. The `id` arguments provide access to retained candidates without changing the original fitted object. Refits for candidate extraction use the requested estimation method and are performed when needed, rather than fitting every retained candidate for full inference during the search.

For `method="moments"`, selection requires fixed-only ordinary or weighted regression with `cov=~units`. For `method="ML"` or `method="REML"`, supported univariate random effects and residual covariance structures may remain in the model. Only fixed regression columns are selected. The random and residual structures, supplied covariance matrices and observation set remain unchanged across candidates, while unknown covariance parameters are re-estimated for each candidate.

For likelihood-based fits, fixed-effect subsets are compared using ML likelihoods. In particular, a call with `method="REML"` fits the specified full model by REML but switches to ML for subset comparison. Both methods are identified in the printed output and stored result. REML likelihoods are not used to rank models with different fixed-effects designs. A subsequent ordinary `st.lmm` call can fit the user's chosen formula by REML.

The following named components are accepted in `selection`:

`search` One of `"exhaustive"` (default), `"forward"`, `"backward"` or `"seqrep"`. Exhaustive search examines all allowed subsets. Forward selection adds a source column at each step; backward selection removes one. Sequential replacement improves visited subsets by exchanging source columns.

nvmax Largest number of selected original data columns; default 8, limited by the available columns. The intercept does not count towards this limit. Zero requests the model without any candidate source column, retaining the intercept when it is part of the fixed formula. Ordinarily the smallest reported model contains one more selected column than the forced-in set. If nvmax equals the number of forced-in columns, the model containing exactly that set is retained instead; this also permits nvmax=0 with an empty forced-in set.

nbest Integer from 1 to 10000 giving the number of best candidates retained for each combination of source-column count and estimable fixed-effect rank; default 1. Retaining separate ranks prevents a wider factor from displacing a candidate preferred by a penalized criterion. A comparison summary can therefore contain more than nbest rows for a given source-column count.

nested Logical; defaults to nbest==1. For forward and backward search, TRUE reports the single nested path, using one retained path model at each size regardless of nbest. With FALSE, forward search also considers visited additions and initial source-column prefix models; backward search considers initial and changing path-prefix models.

force.in, force.out Source-column names, indices in object$selection$groups, or logical vectors with one entry per group, identifying columns that must be included or excluded. They cannot overlap. These controls refer to complete data columns, not individual factor levels or contrast columns. Intercept presence is controlled by the fixed formula.

really.big Logical; default FALSE. Explicitly permits an exhaustive search whose estimated candidate count exceeds the large-search threshold: 100000 for ordinary regression or 10000 for a fitted covariance model. It raises the default max.models limit but does not remove that limit.

max.models Maximum number of candidate evaluations; default 100000, or 10000000 when really.big=TRUE. Ordinary-regression reference evaluations count towards this limit: one for the full model, or two when force.out distinguishes the allowed and requested full models. Intermediate reduced-model evaluations used to score sequential replacements also count towards the limit. Heuristic searches cache at most 1000000 distinct candidates without reducing the requested evaluation limit. An exhaustive request exceeding the evaluation limit is rejected before the search; a heuristic reaching its limit is reported as incomplete.

criterion Optional criterion preference for the comparison summary, supplied as a character vector. NULL uses all applicable criteria. All applicable fit statistics are calculated even when a preference is given. This preference controls the summary view and criterion winners; it does not change the greedy search path, select or replace the final fitted model.

Ordinary regression candidates are retained by residual sum of squares within each source-column count and fixed-rank class. Candidates with fitted covariance structures are retained by ML log-likelihood within these classes. Forward, backward and sequential replacement searches are heuristic and do not guarantee the globally best subsets. Their paths are not stopped merely because a penalized criterion worsens. Exhaustive search is complete only if all required candidates were evaluated successfully; fit failures and an incomplete search are reported. Covariance optimization convergence is checked separately from coverage of the subset search. The retained candidate table is limited to 5000000 slots over combinations of source-column count, fixed-effect rank and nbest; requests exceeding that implementation limit must reduce nvmax or nbest.

Subset summaries accept the following additional arguments through ...:

criterion NULL uses selection$criterion when supplied, otherwise all applicable criteria. A character vector selects one or several criteria from "rss", "rsq", "adjr2", "Cp", "BICm", "AIC", "BIC" and "logLik". Only the last three apply to fitted covariance models.

The aliases "RSS", "r2", "R2", "cp", "aic" and "bic" are also accepted. The summary contains the union of the best requested candidates per source-column count for each criterion. Consequently, different criteria can contribute different rows at the same size. A requested criterion that is undefined for every retained model produces an error; the automatic criterion set leaves such a winner unavailable.

`all.best` Logical; TRUE (default) uses the search's `nbest` for each requested criterion and size. FALSE uses one best model per criterion and size. It does not change the stored search results.

`matrix` Logical; TRUE (default) constructs the summary's inclusion table `outmat`. FALSE leaves that component NULL.

`matrix.logical` Logical; FALSE (default) represents inclusion in `outmat` with stars and blanks. TRUE constructs a logical matrix. The `which` component is always logical and includes the intercept column when fitted; `outmat` omits that intercept column.

`DF` Optional positive scalar for ordinary regression. It replaces $n - I$ in the usual summary formulas, where I is 1 when an intercept is fitted and 0 otherwise. The corresponding effective sample count is $DF + I$. This changes adjusted R^2, C_p and `BICm`; it does not refit any candidate or change the actual-sample ML log-likelihood, AIC or BIC. NULL uses the observed sample count.

All candidates use one common analysis sample determined from the complete specified model and weights. Missing or non-finite responses, predictors, grouping values or weights are excluded consistently, and zero-weight rows are excluded. Removing a candidate predictor does not add rows back to the sample. A separate refit supplied with the original data can have a different sample if its reduced formula no longer refers to a missing predictor. Reproduction of a comparison score requires the same analysis data. The prepared analysis data are retained in `object$data`; `selection$rows` records the corresponding original row indices.

For ordinary regression, the comparison includes RSS, R^2, adjusted R^2, Mallows' C_p, `BICm`, AIC, BIC and the ML log-likelihood. Here n is the common number of observations, r is the estimable fixed-effect rank including any intercept, RSS is the residual sum of squares and TSS is the corresponding total sum of squares. `BICm` is the normalized regression statistic

$$BICm = n \log(RSS/TSS) + r \log(n).$$

This reproduces the ordinary-regression statistic called `bic` by `regsubsets`, without using that package. `BIC` retains the usual likelihood definition. RSS uses the supplied precision weights. For comparisons with an intercept, TSS is the weighted sum of squared deviations from the weighted response mean. Without an intercept, the comparison uses $TSS = \sum_i y_i^2$, without weighting, even when RSS is weighted. This follows the established `regsubsets` normalization for `rsq`, `adjr2` and `BICm`; the sum includes only the common analysis sample, after excluding missing, non-finite and zero-weight rows. This reporting convention does not change the weighted fits, RSS, log-likelihood, AIC, BIC or the full fitted model's own weighted R^2 calculation. Undefined statistics, including C_p when its reference variance is unavailable or nonpositive, are reported as unavailable. Fitted covariance models report ML log-likelihood, AIC and BIC; ordinary-regression C_p, adjusted R^2 and `BICm` are not assigned a mixed-model interpretation.

The C_p reference follows the `regsubsets` compatibility convention, including when `force.out` is supplied. The quantity $RSS_{\text{requested}}$ is the residual sum of squares of the full specified fixed formula, before applying `force.out`, and r_{allowed} is the estimable fixed-effect rank of the model containing every allowed source column after applying `force.out`. Then

$$s^2_{C_p} = RSS_{\text{requested}}/(n - r_{\text{allowed}}), \qquad C_p = RSS/s^2_{C_p} - n + 2r.$$

Thus the numerator and residual degrees of freedom can come from different models when columns are forced out. This is a compatibility convention; with no forced-out columns, both refer to the same full model. The reference models are not limited by `nvmax`. For `summary(DF=...)`, the effective sample count `DF+I` replaces n in both formulas.

The stored components `full.allowed.rss` and `full.allowed.rank` describe the allowed full model; `full.rss` and `full.rank` retain the same meanings. The components `cp.reference.rss`, `cp.reference.rank` and `cp.reference.DF` record the requested-full RSS, allowed-full rank and residual degrees of freedom actually used for C_p. The `full.allowed.*` and `cp.reference.*` components are also available in the comparison summary, where `cp.reference.DF` reflects any summary `DF` override.

The comparison stores inclusion indicators in `which`, source-column names in `groups`, and retained formulas in `formulas`. Its `statistics` data frame contains `id`, `size`, `rank`, `parameters`, `rss`, `rsq`, `adjr2`, `cp`, `BICm`, `AIC`, `BIC` and `logLik`. Here `size` counts selected data columns, `rank` counts estimable fixed coefficients, and `parameters` includes estimated covariance parameters for the likelihood criteria. Summary objects also provide the statistic vectors directly and a `winners` data frame with the criterion, model identifier and value of each preferred retained model. Model identifiers denote retained rows, not model sizes. Plotting displays an inclusion table ordered by the requested criterion. A criterion can identify different preferred models without rerunning the search because rank-specific candidates are retained. Cross-validation, folds and resampling remain the responsibility of the calling R code; the package supplies fitting and prediction.

Search diagnostics include `evaluated`, `failed`, `complete` and `optimal`. `complete` means that the chosen search procedure finished without failed candidate fits or a reached evaluation limit; `optimal` additionally requires exhaustive search. The `failures` component stores inclusion patterns and messages for the first 100 failed fits. The full count remains in `failed`.

Moment estimators: With `method="moments"`, fixed-effects models use least squares; multivariate fixed-only models with one common fixed design also provide MANOVA tests. Mixed designs require a balanced orthogonal layout. Predefined designs retain their original default calculations. Other balanced orthogonal crossed or nested categorical designs use the general Cornfield–Tukey/Underwood path. Sources without an available exact or quasi-F test are retained with status `"no exact test"` and unavailable F and p values. Quasi-F tests are enabled by default in the general calculations and use sums of two independent mean squares on each side and Satterthwaite DF for both sides. Unsupported nonorthogonal layouts and unresolved covariance inference raise errors; no likelihood fit is substituted. Linear functions combining independent error strata use Satterthwaite DF by default; this does not change the design-specific exact ANOVA F tests. Negative variance-component estimates are retained without truncation. Certified balanced categorical layouts use centred grouped projections for source sums of squares and sums of products, with analytic EMS coefficients. Integer cell-count checks verify the source intersections and orthogonality; equal marginal factor counts alone are insufficient. This calculation keeps the same moment equations, constrained interactions and inference methods. The numerical orthogonal-basis calculation remains available for layouts or rank tolerances outside that certificate. Mixed categorical fits and fixed-only fits using `control$design="general"` record the selected calculation in `object$moments$calculation`. The stored estimation details are in `object$moments`, including residual sums of squares, mean squares and, where applicable, variance components and expected mean squares (EMS). Printing shows EMS in a separate section immediately after the residual section. Displayed formulas use `s2` and labels such as `EMS(Intercept)`. Full factor names are retained when the line fits; unique short labels and an abbreviation key are used only for expressions exceeding `char.per.line`. EMS equals signs are aligned. EMS component columns

are also aligned when the complete layout fits; otherwise only the equals signs are aligned. Test columns are displayed in the order DF1, DF2, SS, MS, F, p, where present. The numerical test and residual tables retain their full stored EMS expressions. Directly extracted test tables, MANOVA tables and numeric estimation components use the common compiled printing controls, including digits, top, bottom and char.per.line. Moment fits support control$rank.tol and control$definite.tol, plus the categorical analysis options below. Likelihood, AIC and BIC are unavailable.

design="auto" Predefined paths are retained, with automatic fallback for other balanced orthogonal designs. design="general" selects the general path for univariate fits and multivariate mixed fits. Multivariate mixed fits use common complete observations and a common fixed design for the traits. An intercept and positive residual DF are required. Factors must have complete observations and balanced observed groups. Fixed nested factors, random regressions, relationship matrices and nonorthogonal sources are unsupported.

quasi.f=TRUE A quasi-F search is performed by default when no single exact denominator exists. The EMS under the null must match on both sides. The actual source DF are stored in sourceDF, the approximate numerator DF in DF1, denominator DF in DF2, and mean-square combinations in numerator and denominator. P-values use full precision DF, unlike GAD 2.0 which rounds these DF before evaluation. Unresolved tests remain explicitly unavailable. Exact tests take precedence. FALSE requests exact tests only.

pool=character(), pool.method="pool" Explicit sequential source names; no automatic selection by p-value. pool.method is one method or one per source. "pool" merges the selected source's SS and DF with its single exact denominator, named Pooled1, etc.; "eliminate" merges them with the residual. Each step removes that source component from the EMS and reconstructs tests. Pooling requires a single exact denominator; elimination accepts an available exact or quasi-F test. Untestable sources cannot be removed. Names in later steps refer to the current table. object$moments$pooled contains the ANOVA, history, components and residual. Multivariate mixed fits apply the same source sequence to all traits and combine the full sums-of-squares-and-products matrices and their DF. The pooled object also contains SSCP, mean.products, and strata. The pooled source matrices are available as st.lmm.components(object, "sscp")$pooled. anova and printing use the pooled hypothesis table. The fitted model, coef, vcov, fitted, residuals and st.lsm retain the original unpooled model and strata, stored in object$moments$anova. Thus pooling does not silently change the hypothesis represented by a subsequent coefficient contrast. Elimination uses the updated residual DF in p-values (GAD 2.0 sometimes uses the original DF).

components=FALSE With TRUE, a components-of-variation table for fixed and random sources is stored in object$moments$components. A tested source's estimate is its test numerator minus denominator mean square, divided by its own EMS coefficient. A source without a test has an unavailable summary estimate, even when the full random-component moment equations are solvable. Random variance estimates from those full equations remain separately stored in variance.components. Negative estimates are retained, their SD is unavailable, and their percentage is zero. Percentages follow GAD: shares of the sum of nonnegative component SDs, not shares of variance. Multivariate mixed fits report these summaries and percentages separately by trait. Pooled summaries are stored in object$moments$pooled$components.

cochran=FALSE With TRUE, Cochran's C, its bounded p-value, observed cell count, replication, largest-variance cell, cell variances and means are stored in object$moments$cochran. Cells are grouped by the factors in the model. Equal replication of at least two observations per cell is required. Multivariate mixed fits use the same grouping for every trait and return

a named list with one diagnostic per trait. Each p-value uses the GAD/Underwood F-based approximation, capped at one.

`quasi.f=TRUE` and explicit pooling select the general path. Component summaries and Cochran's C use the selected fitting path. Mean comparisons continue through `st.lsm`; SNK is not implemented.

In a nested-model comparison, differences and test statistics for the first (reference) model are displayed as blank cells. Their stored values remain numeric `NA`. The exact F tests use the residual mean square and denominator DF of the largest model in the comparison. The table reports these denominator DF in `DF2`. With `test="none"`, the comparison reports changes in rank and residual sums of squares without F tests.

Replicated regression uses the ordinary fixed formula and sequential tests: `st.lmm(y ~ x + factor(x), data=d, method="moments", type=1)`. The numeric term must precede the factor term. In `anova(m)`, the `x` row tests the linear trend and the `factor(x)` row tests lack of fit; the residual is pure error. With g distinct predictor values and n observations, these have 1, $g-2$ and $n-g$ DF, respectively. Both F tests use the pure-error mean square. Replication and positive pure-error variance are required for inference.

Fixed-only moment fits use the design rank for residual and hypothesis DF. Redundant columns, including those in `x + factor(x)`, are retained in the coefficient coordinates and handled by a pivoted QR basis. Linear functions are checked for estimability. Aliased coefficients are reported as `NA`; a term with no estimable hypothesis has zero numerator DF and unavailable F and p values. Predefined mixed moment designs still require a full-rank fixed design. Type I tests include the sequential intercept contribution before all predictor terms. Its sum of squares is $n\bar{y}^2$; its F test uses the fitted model's residual mean square, as do the later sequential terms. This tests the overall mean and differs from an adjusted intercept test. When every raw numeric coefficient is non-estimable, the model printout omits the regression coefficient table; the stored `regression` component retains the coefficient estimability results.

Multivariate mixed moment fits use common complete observations and the same categorical fixed design for every trait. Ordinary categorical random intercepts and `cov=~units` are available for balanced orthogonal designs. The general path constructs a common set of strata from the factor structure. Each trait's mean squares determine its exact or quasi-F test. One example is the multi-environment layout `random=~gen+env+gen:env+rep:env`, with genotypes crossed with environments, equally replicated blocks within each environment, and one observation per genotype/environment/replicate cell. Labels of replicates are common across environments. With `fixed=cbind(y1, y2) ~ gen` and `random=~env+gen:env+rep:env`, the same design estimates least-squares means for each genotype and trait. `st.lsm(fit, ~gen)` returns their estimates, SE, DF and confidence limits. With `random.test="F"`, the general path uses exact denominators where available and searches for quasi-F tests for the remaining terms.

Each orthogonal source gives a sums-of-squares-and-products (SSCP) matrix. Dividing by source DF gives a mean-product matrix. The engine solves the expected mean-product equations for every variance and covariance entry using one common coefficient matrix, without likelihood optimization. For the multi-environment layout with random genotypes,

$$\widehat{\Sigma}_G = (MP_G - MP_{G:E})/(er),$$

where e and r are the numbers of environments and replicates. The same equations apply to the diagonal variances and off-diagonal covariances. All raw component matrices are retained, including negative variances or indefinite matrices. `object$moments$positive.semidefinite` reports their numerical semidefiniteness; no matrix is repaired or truncated.

`st.lmm.components(object, "sscp")$strata` and `object$moments$mean.products` are named lists of trait matrices for every source and `Residual`. `object$moments$strata` gives source DF; `object$moments$EMS` contains expected mean-product coefficients; `object$moments$moment.coefficients` contains the square system used to estimate the components. The estimates are available as `object$engine$sigma`, where the ordinary residual component is named `units`. The moments-specific diagnostic record `object$moments$covariance.components` retains the stratum name `Residual`. Printing shows the strata, mean products, coefficient matrix and all covariance component matrices automatically. `anova` returns cached trait-wise fixed or requested random F tests. `object$moments$stratum.geometry` stores the coefficient matrices that weight each mean-product stratum in the fixed-coefficient covariance. `st.lsm` uses this information for within-trait confidence intervals and tests. Joint tests in `st.lsm` concern one trait per hypothesis. Multivariate prediction and random-effect BLUPs use likelihood fits.

For a marker dataset, `y` is a snapshot of `gs.get.y(data.set, names = TRUE)` and `gen` is a factor of its individual identifiers, in marker-data order. Missing performance values are retained as `NA` and omitted from estimation by the ordinary model machinery. With `data="maize", ucov=list(gen="A")`, the quoted `"A"` requests the internally constructed VanRaden relationship matrix, with `coding="0 1 2"`. It is constructed on demand and reused within the call. All marker individuals enter the relationship matrix, including those without observed responses. No caller object is created or changed.

Only quoted strings are symbolic covariance requests. Unquoted `A` evaluates the actual R object named `A`, including when `data` names a stored data set. Prepared lists such as `U <- list(gen="A")` work identically to inline lists. Symbolic covariance requests require a stored marker or hybrid data-set input; ordinary data frames require actual numeric covariance matrices.

For stored hybrid data, `ucov=list(mother="A", father="A", "mother:father"="kro(a,a)")` requests parental GCA matrices and their mapped SCA product. The list names correspond to the actual parental factor names. The quoted product `"kro(A,A)"` is equivalent; the placeholders refer to the two parents in term order. The entire product is quoted, not only its operands. No R function named `kro` is needed. Other symbolic products are rejected; known general products require an actual `kronecker(M1,M2)` matrix. Strings are never evaluated as R code.

Successful converged top-level fits using quoted symbolic covariance requests also store fitted prediction state in their source marker datasets. Hybrid parental sources share that state. It is accessible through `st.marker.results` and `st.predict`. Failed fits and internal refits do not replace it. Numeric covariance matrices do not establish a new marker association. Simulation populations do not receive this state.

This interface fits the genomic second stage when stored performance values are genotype means or adjusted means. It does not fit a field-trial first stage or calculate first-stage weights. `st.load.performance.data` loads those responses, and `gs.set.performance.data` replaces them. Without `ucov`, genotype effects remain independent and no relationship matrix is constructed. The fitted object's `data` and `ucov` retain the actual model data and covariance matrices, so subsequent dataset changes do not alter an existing fit.

Fixed formulas support `+`, `-`, `:`, `*`, `/`, powers, `0`/`1`, parentheses and numeric expressions such as `I(x^2)`. The formula compiler walks R's parsed formula objects; it does not parse deparsed strings. Numeric transformations are evaluated in the formula/data environment. In a fixed formula, `.` expands to the other named data columns after excluding response columns. In mixed models, grouping-only identifiers and residual-structure columns are excluded from this automatic expansion; explicitly naming such a column in the fixed formula still includes it. Fixed-formula `offset(...)` terms and the optional argument offset contribute known values to the response mean.

They are summed once, retained through refitting and updates, and reused for prediction according to the offset rules above. Unordered factor labels that are all finite numbers are sorted numerically; ordered levels retain their declared order. Factor coding follows the explicit `contrasts` argument, then a factor's contrasts attribute, then the global contrasts option, including its separate setting for ordered factors. The fitted coding is retained for prediction. Type III hypotheses use sum coding internally. Type II containment requires matching continuous variables and nesting of categorical variables.

For a multivariate response, genetic trait covariances for a random `gen` term are available as `m$engine$sigma$gen`. `cov2cor(as.matrix(m$engine$sigma$gen))` returns the corresponding genetic correlation matrix. These are correlations between random genetic effects; they are distinct from correlations between observed traits.

For a multivariate response, every bare random term and `units` defaults to an unstructured trait covariance, equivalent to `us(trait)`. `diag(trait):gen` and `diag(trait):units` specify independent traits. Each `at(env):gen` stratum receives its own block; `at(env,"local"):gen` selects levels. A multivariate random regression has a full covariance over trait by regression coefficient unless constrained explicitly.

For `random = ~gen` with `ucov = list(gen = A)`, the genetic effects have covariance $\sigma_g^2 A$; the variance multiplier is estimated by ML or REML. The supplied matrix is the relationship matrix itself; an inverse is not expected. No centring or rescaling is applied to the supplied matrix, so the reported variance component is a multiplier of its original scale. Singular genomic relationship matrices are supported. For multivariate effects or random regressions, the covariance between the coefficient vectors of groups g and h is $A_{gh}G$, where G is the estimated trait/regression covariance.

Names in `ucov` match the grouping part of a random term: `gen` also applies to `~Time | gen`, `us(trait):gen` and each independent stratum of `at(env):gen`. An interaction grouping term such as `gen:env` requires its own matrix and list name `"gen:env"`; a matrix named `gen` is not automatically expanded over environments. Residual covariance remains specified through `cov`.

All levels in a supplied matrix are retained in `object$engine$u`, including levels without phenotypic records. Conditional predictions for these levels use their relationship-based BLUPs. Levels absent from the supplied matrix are rejected because their relationships have not been specified; `allow.new.levels` does not supply missing relationships. The aligned matrices are retained in `object$ucov`; changing or removing the original matrix after fitting does not change predictions.

Missing response cells are retained as missing cells. A row with missing model predictors or grouping values is excluded for all its traits. Without selection or observation weights, returned fitted values and residuals are restored to the original row layout. Selection and weighted fits retain the common analysis sample in the fitted object; the `observation.rows` attribute maps those rows to the supplied data. Non-estimable fixed functions are returned as unavailable. Fitted values and residuals are matrices with one row per observation and one column per trait. Coefficient vectors are named.

The default requested fixed-effect inference is a Type III F-test with Satterthwaite denominator degrees of freedom. Certified balanced orthogonal terms use exact F ratios when available; other terms retain Satterthwaite inference, as in **lmerTest**. Satterthwaite inference uses the observed covariance-parameter information and derivatives of fixed-effect covariance. Tests are computed separately by trait and jointly across traits. Zero independent random-variance components use the first-order standard-deviation information approximation; singular nonnegative information uses its positive-eigenvalue generalized inverse. These cases are identified in the engine information status. Negative-curvature or nonconvergence cases retain unavailable inference rather than substituting

another degrees-of-freedom method. Asymptotic tests are Wald tests and display infinite denominator DF. Kenward–Roger uses expected REML information in linear covariance-component coordinates, adjusts the fixed-effect covariance, and applies hypothesis-specific denominator DF and F scaling. Nonconverged or unidentified covariance models do not yield Kenward–Roger inference. `st.lsm(object, L=L, joint=TRUE)` provides explicit hypotheses and the scaling factor. The raw engine covariance is retained for estimation; `vcov` and `st.lsm` expose the adjustment.

With `random.test="auto"`, a certified balanced orthogonal design uses exact random-effect F tests when every random term has one exact denominator; otherwise all random terms use LRT. Explicit `random.test="LRT"` removes each whole random term in turn and refits the reduced model with the same observations, fixed effects and estimation method. The statistic is twice the full-minus-reduced log-likelihood; the chi-squared degrees of freedom are the difference in the number of free covariance parameters. REML fits use REML likelihoods. This corresponds to whole-term deletion in `lmerTest::ranova(reduce.terms=FALSE)`, including joint deletion of all strata and coefficients of a random regression or multivariate term. It does not perform stepwise slope reductions. The chi-squared reference distribution is approximate, without a boundary-mixture correction; the optional bootstrap provides an alternative when that approximation is unsuitable. LRT tests use no simulations and do not consume random numbers. A failed reduced fit or an invalid likelihood comparison gives unavailable inference with a reason.

Bootstrap random-term tests refit full and reduced models with identical fixed effects, using the fitted method (ML or REML), simulated under the reduced model. They use R's random-number generator; `set.seed` controls repeatability. All strata and trait/regression coefficients belonging to a random term are tested together. The p-value is `(exceedances+1)/(replicates+1)`. Monte Carlo standard error and refit failures are reported; any failed refit causes the p-value to be withheld. Bootstrap may be expensive: `random.test="none"` is useful for exploratory fitting. Printing never simulates or refits.

Marginal and conditional R-squared use the sample variance of fitted fixed effects and the observation-averaged diagonal of random/residual covariance, separately by trait over observed responses. RMSE uses conditional residuals. `AIC` and `BIC` dispatch through the log-likelihood method; the counted parameters are the estimable fixed coefficients plus estimated covariance parameters. Usual restrictions on comparing REML likelihoods across different fixed models apply.

New fits retain fixed-effect and random-regression prediction recipes for orthogonal polynomials, scaling and spline bases, including their training coefficients, center, scale and knots. Prediction and least-squares means reuse these recipes. Transformations are calibrated before row subsetting and missing-value removal. Older saved fits without the required metadata must be refitted or use precomputed columns. Formulas may call user functions, so their environments must remain available to evaluate new data. All fitted values, tests and statistics can be serialized without external pointers.

Prediction of selected random effects and uncertainty: The following extensions apply to ML and REML fits. The default `predict(object)$fit` contains the marginal fitted means, including offsets. Conditional prediction adds the selected fitted random coefficients. Random-only prediction returns those random contributions and excludes fixed effects and offsets. Formula selectors can select an intercept or slope within a correlated block. Selection changes the prediction target only: all fitted random components remain in the observation covariance used for conditioning. A named-block selector therefore differs from deleting terms and refitting.

The target fixed design is X_*; C_* is the prior covariance of its selected random contribution, K_* is its covariance with the observed responses, and V is the full observed-response covariance. With

estimated fixed effects, the latent-mean prediction-error covariance is

$$C_* - K_* V^{-1} K_*' + D \operatorname{Var}(\widehat{\beta}) D', \qquad D = X_* - K_* V^{-1} X.$$

Random-only targets use $X_* = 0$; marginal targets use $C_* = K_* = 0$. `se.fit` returns the square roots of the diagonal. This includes the joint fixed/random prediction-error correction and is not obtained by adding separate coefficient standard errors. Covariance parameters are held fixed, future observation residuals are excluded, and no covariance-parameter uncertainty is implied. Nonestimable fixed combinations are returned as unavailable.

For independent unseen groups allowed by `allow.new.levels=TRUE`, the random mean is zero and its prior variance contributes to prediction error. For named levels already present in a supplied relationship matrix, relationship-based prediction and uncertainty use the covariance with the observed levels, even when the level had no observed response. A new covariance stratum has no estimated variance, so its conditional standard error is unavailable and requesting one produces an error. New data for a direct observation kernel require an explicitly defined target kernel and are not supported by this interface.

`st.lmm.components(object, c("fixed", "covariance"))` provides the physical parameter layout accepted by `newparams`. Supplying covariance parameters performs one covariance evaluation and GLS/BLUP solve without optimization. If `fixed` is omitted, fixed coefficients are estimated at the supplied covariance and their estimation contributes to prediction error. If `fixed` is supplied, its values are treated as known and the random conditional mean is recomputed from the residuals relative to those values. Its conditional covariance is $C_* - K_* V^{-1} K_*'$; the fixed-effect estimation term is omitted. Accordingly, supplying the fitted coefficient values explicitly gives the same point prediction but can give a different standard error. Supplying complete fixed coefficients also defines predictions involving columns aliased during fitting. The fitted object is not modified.

The returned uncertainty list has attributes describing its interpretation, whether fixed coefficients are treated as known, the selected coefficients, and the exclusion of future residual noise. With `na.exclude`, training-row predictions are padded at omitted positions; missing-response prediction rows retained under the default policy receive predictions. The `re.form` and `newparams` extensions are not available for `method="moments"`; its existing marginal/conditional prediction and interval interface is unchanged.

Value

`st.options` returns a named list of the ten current printing options: visibly for queries with no argument or `NULL`, and invisibly after an update supplied as a list. The model-fitting function returns a serializable S3 object of class `st.lmm`, with the call/formulas/data, compiled `engine` fit, compiled `model` metadata/designs, cached `tests`, numeric fixed-regression `regression` table, per-trait `statistics`, and inference controls. The engine includes coefficient covariance, covariance blocks, random-effect predictions, alias/estimability maps and parameter-information derivatives for subsequent linear-function inference. When subset comparison is requested, the object also contains `selection`, holding source-column groups, retained models, comparison statistics, search settings and diagnostics. Its default summary has class `st.lmm.selection.summary`. The full-model fit is not replaced by a selected candidate.

Model data display

The model description begins with `Data:`, followed by `Fixed:`. The data argument recorded in the fitted call identifies the input without evaluating it again. A fit without a data argument reports the

formula environment, or a series of earlier fits when those provide the model data. An embedded data snapshot reports stored data. Earlier objects without a recorded call report that the source was not recorded.

Fixed-effect table display

The response-label column `trait` is retained in stored fixed-effect test tables. Model printing, `anova` printing and extracted test-table printing omit this redundant column when the table contains only one trait. Tables containing several traits retain it. This display choice does not change the stored results or their numerical values.

Covariance formula syntax

The following formulas are interpreted by the `st.lmm` model compiler. The symbols `trait` and `units` are reserved in these covariance terms; no columns with these names need to be added to `data`. `units` identifies the original data rows. In terms containing `units`, different rows have independent residuals, while traits measured on the same row may have residual covariances. The separate `occasion | group` form described below permits residual correlation between repeated observations in different rows.

Trait covariance and independent strata:

Term	Meaning
`us(trait)`	Unstructured covariance across traits.
`diag(trait)`	A separate variance per trait; zero trait covariances.
`homdiag(trait)`	A common residual variance; zero trait covariances.
`cs(trait, hom=TRUE)`	A common residual variance and correlation.
`cs(trait, hom=FALSE)`	Separate variances and a common correlation.
`ar1(trait)`	A common residual variance and AR(1) correlations.
`ar1(trait, hom=FALSE)`	Separate variances and AR(1) correlations.
`at(loc)`	A separate, independent block for each location.
`at(loc, "HOH")`	Only the block for location HOH.
`at(loc, c("HOH", "EWE"))`	Two separate location blocks.

For a matrix response such as `cbind(GYLD, TKW, PLH)`, omitting the trait modifier means `us(trait)`, both for random effects and residuals. Thus `~units` and `~us(trait):units` specify the same residual model. An unstructured three-trait block has three variances and three covariances; it does not constrain the traits to have the same variance. `diag(trait)` also gives each trait its own variance, but fixes all off-diagonal covariances to zero. These restrictions apply to the component in that term, not to other random or residual components.

`at(loc)` changes which parameters are shared between locations; `diag(trait)` changes which traits may covary. Combining them as `diag(trait):at(loc):units` estimates one residual variance for every trait by location combination. Selecting several levels with `at()` still gives a separate block for each selected level; it does not pool them into a common variance group.

Single-trait residual formulas: For a single numeric response, each of the following independent-residual blocks has one variance. The counts L and E are the numbers of observed locations and environments, respectively, with no missing grouping values.

Formula for cov	Residual variances	Count
`~units`	One common variance.	1
`~at(loc):units`	One per location.	L
`~at(env):units`	One per environment.	E

The meaning of `env` depends on how that variable was constructed; its name alone does not define an environment or imply a location.

Correlated residuals for repeated observations: For a univariate ML or REML fit, `cov = ~ar1(0 + Visit) | Subject` defines AR(1) residual covariance between visits within each subject. Different subjects have independent residuals. `Visit` must be an R factor; its declared levels, including unobserved levels, determine the occasion order. Both ordered and ordinary factors retain that order. The coefficient formula must consist of this single factor with no intercept, as in `0 + Visit` or `Visit - 1`. Numeric regressions, character vectors, additional predictors and contrast bases are rejected. Each residual stratum may have at most one observed row per subject and visit. Missing responses and incomplete sets of visits are allowed.

With q occasion levels, the following modifiers are supported on the left of `| Subject`:

Modifier	Residual covariance	Parameters
`us(0 + Visit)`	Unstructured.	$q(q+1)/2$
`diag(0 + Visit)`	Separate variances; zero covariances.	q
`homdiag(0 + Visit)`	Common variance; zero covariances.	1
`cs(0 + Visit, hom=TRUE)`	Common variance and correlation.	2
`cs(0 + Visit, hom=FALSE)`	Variances differ; correlation shared.	$q+1$
`ar1(0 + Visit)`	Common variance; AR(1) correlations.	2
`ar1(0 + Visit, hom=FALSE)`	Separate variances; AR(1) correlations.	$q+1$

With one occasion, each structure has one variance and no correlation parameter. The default is `hom=TRUE` for `ar1()` and `hom=FALSE` for `cs()`. For visits at positions i and j, AR(1) specifies residual covariance $s_i s_j \rho^{|i-j|}$; compound symmetry replaces the correlation by ρ for distinct positions. Homogeneous structures use a common s_i. Correlation bounds and variance constraints are the same as in 'Random-coefficient covariance structures'.

AR(1) lags refer to factor-level positions, not differences between numeric labels. Intermediate occasions in the factor levels represent intended gaps. For example, levels `c("1", "2", "3")` retain a lag of two between occasions 1 and 3 even if occasion 2 is unobserved. Homogeneous structures can retain entirely unobserved occasion levels. A separately estimated occasion variance requires observed information in that stratum; otherwise the fit is rejected. A missing level is never silently dropped to shorten AR(1) distances. Insufficient information for a correlation is reported by the covariance-information diagnostics. When all observed within-group pairs have even lags, the likelihood cannot distinguish positive from negative AR(1) correlation; its sign is unidentified, and the profile inventory does not offer an ordinary interval for signed ρ. If some heterogeneous AR(1) variances are zero and no observed pair has two positive standard deviations, the correlation can be unidentified while still affecting directions that increase a zero variance. The optimizer checks these directions. If the remaining boundary curvature cannot be certified, it reports nonconvergence rather than accepting that boundary.

The grouping side can include interactions and independent strata, for example `~ar1(0 + Visit) | at(Treatment):Subject`, which estimates a separate covariance per treatment. Lists add residual components. The existing `~at(Treatment):units` notation continues to describe independent residuals with a treatment-specific variance. With precision weights, covariance between rows i and j is divided by $\sqrt{w_i w_j}$. Random effects and `ucov` may be used alongside these residual structures.

Refitting, simulation and construction of future residual covariance retain the fitted occasion levels and their order; an unknown occasion level is an error. Mean predictions retain their documented fixed/random-effect meaning; they do not add a fitted residual process to the predicted mean. These structures require ML or REML. The inference restrictions described for the corresponding covariance families also apply to residual blocks. Correlated residuals use the exact block or dense covariance backend; forcing an incompatible sparse backend is an error. Model comparisons report likelihood and information criteria, but do not infer an LRT reference distribution for changes in the residual covariance family.

Separable spatial residual covariance: For a univariate ML or REML fit, `cov = ~ar1(Row):ar1(Column)` estimates a homogeneous separable AR(1) by AR(1) plot-error covariance. The colon denotes a direct product, using the formula style of ASReml-R 3. The existing `cov` argument continues to specify the residual covariance. There are three estimated parameters: one residual variance and one correlation for each factor. For two plots in rows r, r' and columns c, c', their residual covariance is

$$\sigma_e^2 \rho_r^{|r-r'|} \rho_c^{|c-c'|}.$$

Both variables must be distinct R factors, each with at least two declared levels. Declared factor levels determine the positions and spacing, including unobserved intermediate rows or columns. Data rows may be in any order; the first factor varies fastest in the retained coefficient covariance matrix. For the formula above, rows within columns give

$$R = \sigma_e^2 \{A_C(\rho_c) \otimes A_R(\rho_r)\},$$

where $[A_m(\rho)]_{ij} = \rho^{|i-j|}$. Missing responses and incomplete grids are supported. A grid may have at most one observed response per row-column combination. The form `~ar1(Row):ar1(Column) | Field` gives independent fields sharing the same spatial parameters; `at()` strata on the grouping side estimate separate covariance blocks. Factor levels retain their order for refitting and future residual covariance construction. Unknown coordinate levels are rejected.

Random replicate and incomplete-block terms may be retained, for example `random = ~Rep + RowBlk + ColBlk`. Precision weights divide residual covariances by $\sqrt{w_i w_j}$. A list of residual formulas adds components; it does not multiply them. In particular, `list(~ar1(0 + Row) | Column, ~ar1(0 + Column) | Row)` is an additive row-column covariance, not a separable AR(1) by AR(1) covariance.

Analytic covariance derivatives are used for likelihood optimization and Satterthwaite inference. Asymptotic inference is also available. Kenward–Roger inference and likelihood profiles are not implemented for the separable product. Correlations have bounds $[-1, 1]$; singular parameter combinations are admissible only if the full model covariance remains positive definite on the observed responses. With observations only at even lags in a direction, the corresponding correlation sign is unidentified. The optimizer checks nonzero-correlation alternatives before accepting a zero-correlation stationary fit. A zero spatial variance whose two-dimensional covariance cone cannot be certified is reported as nonconverged, rather than accepted as a verified boundary optimum. The printed summary shows the variance and both correlations; the full coefficient covariance remains available from `st.lmm.components()`.

Multivariate residual formulas: For T response traits, $U = T(T+1)/2$ denotes the number of parameters in one unstructured trait covariance matrix. With all traits observed in every stratum, the following are the numbers of free residual covariance parameters; random-effect parameters are additional.

Formula for cov	Residual structure	Count
`~units`	Unstructured trait covariance.	U
`~us(trait):units`	Same unstructured model.	U
`~diag(trait):units`	Separate trait variances.	T
`~homdiag(trait):units`	Common variance; zero covariance.	1
`~cs(trait, hom=TRUE):units`	Common variance and correlation.	2
`~cs(trait, hom=FALSE):units`	Variances differ; correlation shared.	$T+1$
`~ar1(trait):units`	Common variance; AR(1).	2
`~ar1(trait, hom=FALSE):units`	Separate variances; AR(1).	$T+1$
`~at(loc):units`	Unstructured, per location.	LU
`~us(trait):at(loc):units`	Same location model.	LU
`~diag(trait):at(loc):units`	Trait variances per location.	LT
`~at(env):units`	Unstructured, per environment.	EU
`~diag(trait):at(env):units`	Trait variances per environment.	ET

For example, `~diag(trait):at(loc):units` assumes, for unit i at location l,

$$\mathrm{Var}(e_i) = \mathrm{diag}(\sigma^2_{1l}, \ldots, \sigma^2_{Tl}).$$

Using `~at(loc):units` instead replaces that diagonal matrix by a location-specific unstructured matrix. In both cases, different data rows have zero residual covariance. Random effects can still induce covariance between their observations.

The `homdiag`, `cs` and `ar1` trait modifiers require ML or REML and may also be combined with `at()` strata. Their parameter counts above assume at least two traits; with one trait there is one variance and no correlation parameter. For AR(1), the left-to-right response order in `cbind(y1, y2, y3)` defines positions 1, 2 and 3. Changing that order can change the covariance model. Traits on different data rows remain residual-independent. These formulas do not combine a trait covariance with an additional repeated-occasion covariance.

Location versus year by location: the barley MET example: The `barley.met` dataset contains five recorded traits, five locations and three years. The following covariance examples use the three responses `GYLD`, `TKW` and `PLH`. Here `env` is a year-by-location factor:

```
data("v-lmm-barley-met", package = "SelectionTools")
barley.met$env <- interaction(barley.met$year, barley.met$loc,
                              drop = TRUE)
```

There are 15 observed environments. Therefore:

Formula for cov	Residual structure	Count
`~at(env):units`	Full trait covariance per year by location.	90
`~diag(trait):at(env):units`	Trait variances per year by location.	45
`~at(loc):units`	Full trait covariance per location.	30
`~diag(trait):at(loc):units`	Trait variances per location.	15

The location formulas share each location's residual parameters across years. The environment formulas allow them to differ between years within the same location. Equivalently, for these data, `~at(year):at(loc):units` specifies separate unstructured blocks for the observed year by location combinations, and `~diag(trait):at(year):at(loc):units` specifies separate variances for each trait in each such combination.

With eight bare random terms, each term has an unstructured three-trait covariance matrix, giving 48 random-effect parameters in total. Together with `~at(env):units`, this gives 138 covariance parameters. Using `~diag(trait):at(loc):units` gives 63 instead. This is a different residual model; reducing the parameter count does not by itself establish convergence or justify the covariance restriction.

Selected levels and combinations of formulas: An `at()` level selector activates a covariance component only for the named levels; it does not subset the data. For example, residual terms using only `at(loc, "HOH")` do not provide residual variation for observations at the other locations. A fit for HOH alone requires a corresponding data subset; a fit including other locations requires residual terms covering them. The following level labels provide a complete partition of the barley MET locations:

```
cov = list(
    ~diag(trait):at(loc, c("HOH", "EWE")):units,
    ~us(trait):at(loc, c("EIN", "OLI", "HFH")):units
)
```

This estimates three independent trait variances at each of HOH and EWE, and a full three-trait covariance matrix separately at each of the remaining locations: 24 residual parameters in total. A list adds the specified covariance components. Overlapping residual terms require an intended additive covariance model with identifiable parameters.

The same modifiers for random effects: For a multivariate response, the corresponding random formulas are:

Formula for random	Random-effect structure
`~gen`	One unstructured trait matrix for genotype effects.
`~us(trait):gen`	The same genotype model, stated explicitly.
`~diag(trait):gen`	Separate genotype variances per trait; zero trait covariances.
`~at(loc):gen`	Independent genotype effects per location; a trait matrix per location.
`~diag(trait):at(loc):gen`	Independent trait by location genotype components.
`~gen + gen:loc`	Common genotype effects plus independent genotype by location effects.

The terms `gen:loc` and `at(loc):gen` differ: the former shares one trait covariance matrix across genotype by location effects, whereas the latter estimates a separate matrix per location. Adding `gen` also induces genetic covariance between observations of the same genotype at different locations. Without a matching `ucov` entry, distinct grouping levels are independent; with `ucov`, the supplied grouping covariance is combined with the trait covariance as described in Details.

The residual `at(...):units` notation resembles ASReml-R 3 syntax. SelectionTools also supports the direct-sum subset documented below; this does not implement every ASReml-R 4 `dsum` option.

Covariance composition

For univariate ML/REML models, colon-separated factor covariance modifiers specify separable products. Each axis must be a distinct, directly named R factor with at least two declared levels. Supported product factors are `ar1(Factor)`, `ar1(Factor, hom=TRUE)`, `cs(Factor, hom=TRUE)` and `homdiag(Factor)`. A product has one variance multiplier, not one variance per factor. Each AR1 or CS factor adds one correlation; homdiag adds none. CS correlation bounds depend on that axis's number of levels.

Request	Meaning
`cov=~ar1(Row):cs(Column,hom=TRUE) | Field`	Independent fields with shared product parameters.
`cov=list(~ar1(Row):cs(Column,hom=TRUE) | Field, ~units)`	Product covariance plus an independently estimated
`cov=~(ar1(Row):cs(Column,hom=TRUE) | Field)+units`	Equivalent additive formula; parentheses delimit the
`cov=~dsum(~ar1(Row):cs(Column,hom=TRUE) | Env, group=Field)`	Environment-specific product parameters, shared ac
`random=~ar1(Visit):homdiag(Season) | Subject`	A product covariance among random coefficients w

Products can have more than two axes and can be parenthesized associatively. The first axis varies fastest in the retained coefficient covariance matrix. AR1 lags use positions in declared factor levels, including unobserved levels. They are not numeric distances between observation labels. Missing grid cells are supported; their declared levels determine the spacing. Residual products require at most one observed response per cell and group; random coefficients can have repeated measurements in a cell. A named `ucov` matrix combines the coefficient product with the known covariance between grouping levels. Precision weights scale the whole observation covariance by the reciprocal square roots of the weights.

`dsum` accepts exactly one unnamed one-sided formula `dsum(~covariance | stratum)` and optional `group=Subject`. The stratum and optional group must be distinct directly named data factors. For example, `dsum(~units | Env)` equals `at(Env):units`, and `dsum(~ar1(0+Visit) | Treatment, group=Subject)` equals `ar1(0+Visit) | at(Treatment):Subject`. Existing within-row multivariate trait covariance also works with `dsum(~units | Env)`. The constructor is residual-only. Independent random strata are specified by `at(stratum)` in the grouping expression. Unknown named arguments, nested dsum, repeated axes and malformed formulas are errors. A list of formulas specifies additive covariance components.

Heterogeneous product factors (`hom=FALSE`, `diag`) and arbitrary unstructured product factors are not implemented. The supported `us(trait)` exception for multivariate repeated records is described below. CS product factors must explicitly request `hom=TRUE`; their ordinary non-product default is unchanged. The fitted covariance layout, axis names, bounds and parameters are retained for refitting, prediction, simulation and `st.lmm.objective`. User-supplied physical covariance matrices must obey the fitted product shape, not merely be positive semidefinite. Separable families support Satterthwaite and asymptotic inference; Kenward–Roger and likelihood profiles are rejected explicitly. At a zero product variance with unidentified correlations, convergence is not certified unless its covariance cone can be checked. Existing `ar1(Row):ar1(Column)` residual models retain their established path.

Multivariate repeated records

For a matrix response with at least two traits, one unstructured trait covariance can be combined with one occasion-correlation factor inside an independent residual group:

```
st.lmm(cbind(y1, y2) ~ treatment + x,
    cov = ~us(trait):ar1(Visit) | Subject,
    data = dat, ddf = "asymptotic", random.test = "none")

cov = ~us(trait):cs(Visit, hom=TRUE) | Subject
cov = ~us(trait):homdiag(Visit) | Subject
```

Here `trait` is the virtual response-trait dimension; it is not a required data column. `Visit` must be an R factor with at least two declared levels. `Subject` is one directly named grouping factor. Its distinct levels define independent residual groups that share all trait and occasion parameters. Reversing the two product factors gives the same canonical model and coefficient order.

With p traits and q declared occasions, the coefficient covariance is $\Sigma_T \otimes R_O$. Occasion varies fastest within trait in the returned coefficient covariance. Σ_T is one full $p \times p$ trait covariance matrix. R_O has diagonal one and is AR1, homogeneous compound symmetry, or identity for `homdiag`. The number of estimated covariance parameters is $p(p+1)/2 + 1$ for AR1 or CS and $p(p+1)/2$ for homdiag. There is no second occasion variance multiplier. In particular, homdiag occasions are independent, while traits measured on the same occasion can remain correlated. For AR1, correlation between occasions at declared level positions a, b is $\rho^{|a-b|}$. For CS, all distinct occasions have correlation ρ, whose lower bound is $-1/(q-1)$.

Missing trait responses can be retained independently under the default missing-response handling or `na.action=na.pass`. A missing cell does not require dropping the other observed traits in its row. Whole missing occasions and incomplete subject sequences are supported. The declared factor levels determine the occasion spacing: observed levels 1 and 3 have AR1 correlation ρ^2 even when level 2 has no data row. Explicit row-removing actions such as `na.omit` retain their ordinary whole-row semantics. There may be at most one observed response per subject, occasion and trait; complementary missing-trait rows at the same subject/occasion key are allowed.

The observed cells must identify the requested parameters. The compiled check tests covariance directions at generic interior parameter values, using the fixed-effect projection for REML. Missing cross-trait support, fixed-effect absorption and dependence between random and residual components can cause an explicit error. For example, an unrestricted subject-level trait random intercept together with a CS trait-product residual generally cannot identify the additional occasion correlation. This check concerns local covariance identifiability, rather than a proof of unique global parameter labels in every possible nonlinear mixture.

To certify an AR1 correlation sign, this implementation additionally requires at least one observed odd-lag pair within a trait and subject. An even-lag-only sequence cannot distinguish positive and negative rho. Patterns that would determine the sign only through cross-trait covariance are currently rejected as uncertified. Odd gaps larger than one are supported; fitting probes nonzero correlations to escape a flat starting point at zero. At a locally flat or otherwise rank-deficient information matrix, covariance-parameter uncertainty must be treated as unavailable.

Fitting uses ML or REML and compiled L-BFGS-B. Explicit AI requests are rejected. The default optimizer comparison in `st.lmm.allfit` uses L-BFGS-B for this family. Warm/cold refits, response-unit changes, update, serialization, ordinary prediction and simulation retain the trait/occasion layout. The fitted covariance is exported on the original response scale, including when occasion frequencies differ or a declared occasion is absent. `st.lmm.objective` reports trait log-SDs and transformed partial correlations, followed by `rho.occasion` where applicable. Its trait coordinate row/column indices refer to the trait dimensions; SD scale multipliers convert back to original response units. These optimizer coordinates differ from a vector of physical marginal trait correlations.

Supplied covariance matrices for prediction or simulation must have the full trait-by-occasion dimensions, matching coefficient names and the requested product form. Positive semidefiniteness alone is insufficient. A positive-semidefinite singular product can be simulated; a fitted singular or numerically unresolved boundary does not certify ordinary likelihood inference. Satterthwaite inference requires usable covariance-parameter information; asymptotic fixed-effect inference conditions on the fitted covariance. Kenward–Roger adjustment and covariance profiles are currently unsupported and are rejected explicitly.

This extension is residual-only and contains one virtual `us(trait)` and one ordinary occasion axis per product. Additional axes, random trait-products, other trait modifiers within products, heterogeneous occasion scales, nested/stratified grouping expressions and direct sums of trait-products are not supported. Ordinary supported random-effect terms can accompany an identifiable trait-product residual. Explicit fixed/shared covariance constraints remain unavailable for models containing this estimated US trait family. Multivariate precision weights and offsets retain their existing unsupported-interface errors. Multivariate inheritance from earlier fits is a separate extension and is not implied by this syntax.

Fixed and shared covariance parameters

`constraints` specifies parameter values in original response and coefficient units, independently of internal response or design scaling. Constraints require `method="ML"` or `method="REML"`; `method="moments"` is not supported. The named list accepts exactly `fixed` and `shared`. Selectors must come from the compiled model's inventory:

```
inventory <- st.lmm.objective(fit)$constraints
inventory$parameter
inventory$variance
```

Canonical selectors begin with `sd:random:`, `sd:residual:`, `cor:random:` or `cor:residual:`, followed by the exact block and coefficient names. A homogeneous variance uses `common`; a structured correlation uses `rho`; separable products use `rho.1`, `rho.2`, and the corresponding axis numbers. The `variance` inventory provides `var:` aliases for variance values. For example, `var:random:gen:(Intercept)` fixes a variance, whereas `sd:random:gen:(Intercept)` fixes its standard deviation. Inventory `value` entries use the canonical SD or correlation units, not the variance-alias units. Block names include any retained stratum labels; the fitted names identify the valid selectors.

```
st.lmm(y ~ 1, random = ~gen, data = dat,
    constraints = list(fixed = c("var:random:gen:(Intercept)" = 2)),
    ddf = "asymptotic", random.test = "none")

st.lmm(y ~ 1, random = ~gen + env, data = dat,
    constraints = list(shared = list(c(
        "sd:random:gen:(Intercept)", "sd:random:env:(Intercept)"))),
    ddf = "asymptotic", random.test = "none")
```

Every `shared` entry contains at least two selectors of the same physical meaning. Each group consistently uses SD selectors, variance selectors or correlation selectors. A parameter cannot belong to more than one group. A fixed member fixes its whole shared group; multiple fixed values in that group must agree in physical units. Unknown or ambiguous names, duplicated declarations, nonfinite values, incompatible units and infeasible parameter domains are errors. Correlation groups use

the intersection of their compiled bounds, including dimension- dependent compound-symmetry bounds. Random variances may be fixed at zero where the full marginal covariance remains valid.

Supported estimated blocks are diagonal, homogeneous diagonal, compound symmetry, AR1 and the supported separable products. Models containing an estimated unstructured covariance block, including a US trait-by-occasion product, are currently rejected when explicit constraints are requested. The ordinary unconstrained unstructured interface is unchanged. Fixed known covariance matrices remain fixed inputs.

Constrained fitting optimizes only independent parameters using compiled L-BFGS-B. `optimizer="auto"` selects that path; an explicit AI request is rejected. Starts are reconciled with the fixed/shared declarations before optimization. Explicit objective parameter evaluations must already satisfy the constraints and are never silently projected. With every covariance parameter fixed, only the GLS/BLUP calculation is required.

The engine retains the full optimizer-coordinate vector for reconstruction and reports its `full.count` and independent `free.count` in `engine$constraints`. AIC, BIC and log-likelihood degrees of freedom count independent estimated covariance parameters. The constraint inventory records fixed/free/shared status and the independent coordinate index. Satterthwaite inference uses covariance-parameter uncertainty projected through the reduced parameter mapping; fixed parameters have zero parameter uncertainty. The complete raw information matrix describes directions outside the constraint surface as well and need not have the same definiteness or rank. For constrained fits, `engine$uncertainty$free.information` and `free.vcov` retain the reduced information and covariance matrices in the `free.parameters` order when parameter uncertainty is computed.

Asymptotic and Satterthwaite fixed-effect inference, ordinary prediction and simulation, serialization and response/weight refits retain constraints. `random.test="none"` applies here. Kenward–Roger adjustment, covariance likelihood profiles, automatic random-term deletion, subset selection and model-comparison tests with explicit constraints are not implemented and fail explicitly. Physical `newparams` overrides are rejected for constrained objects; the intended covariance model requires an explicit fit. Optimizer comparisons use L-BFGS-B and retain the same physical constraints for every supported start.

Random-coefficient covariance structures

Both embedded `y ~ x + (1+x || group)` and separate `random=~1+x || group` specify independent random coefficients, equivalent to `random=~diag(1+x) | group`. Parenthesized additive bar terms and lists of random formulas are supported consistently. For factor predictors, independence applies to the expanded coefficient columns in the chosen contrast basis, not to whole multi-column terms. `||` cannot accompany covariance modifiers such as `cs()` or `us()`; `|` supports these modifiers. The operator `||` is not permitted in `cov`.

For a univariate ML or REML fit, modifiers around the coefficient formula specify its covariance within each grouping level. For example, `random = ~ar1(0 + year) | gen` uses one random coefficient for every level of `year`, with an AR1 covariance among those coefficients for each genotype. These modifiers do not change the fixed-effects formula.

Modifier	Covariance	Parameters
`us(...)`	Unstructured covariance.	$q(q+1)/2$
`diag(...)`	Separate variances, zero covariances.	q
`homdiag(...)`	One common variance, zero covariances.	1
`cs(..., hom=TRUE)`	Common variance and common correlation.	2

`cs(..., hom=FALSE)`	Separate variances and common correlation.	$q+1$
`ar1(..., hom=TRUE)`	Common variance and AR1 correlations.	2
`ar1(..., hom=FALSE)`	Separate variances and AR1 correlations.	$q+1$

Here q is the number of random coefficient columns. With one column, each structure estimates one variance and has no correlation parameter. The default is `hom=FALSE` for `cs()` and `hom=TRUE` for `ar1()`; `hom` must be a single literal `TRUE` or `FALSE`. Unmodified random regressions and `us()` retain an unstructured covariance. `diag()` continues to estimate a separate variance for each coefficient.

For compound symmetry, $G_{ij} = s_i s_j \rho$ for distinct coefficients and $G_{ii} = s_i^2$, with $-1/(q-1) \leq \rho \leq 1$. For AR1, $G_{ij} = s_i s_j \rho^{|i-j|}$, with $-1 \leq \rho \leq 1$. Homogeneous structures constrain all s_i to the same nonnegative value. These constraints refer to the original coefficient basis, before numerical column scaling.

AR1 distances are positions in the expanded coefficient basis. They are not distances between numeric labels or observation times. For a factor, `0 + year` specifies one coefficient per level. An ordered factor states the intended order explicitly. An intercept, additional numeric terms or different factor coding changes that basis and therefore changes the covariance model. The coefficient names and their order are retained on `object$engine$sigma`.

For random effects, these modifiers apply to coefficient blocks in univariate likelihood fits. They can be combined with other random terms, structured residual covariance and named matrices in `ucov`. Their residual uses are described in 'Covariance formula syntax'. They are not supported by `method="moments"`. Correlation endpoints and zero variances are allowed; a positive-definite total observation covariance is still required.

Satterthwaite and asymptotic inference are available. At a whole zero covariance block, the first-order standard-deviation approximation drops that block's covariance-uncertainty contribution and its unidentified correlation; the engine information status reports a boundary approximation. Kenward–Roger is available for `homdiag()` and homogeneous compound symmetry. It is also available for the equivalent one- and two-coefficient linear covariance families. Heterogeneous compound symmetry and AR1 structures with more than two coefficients have nonlinear covariance restrictions and raise an explicit error if Kenward–Roger is requested. `ddf="Satterthwaite"` and `ddf="asymptotic"` provide inference for those structures. Nonconverged or unidentified fits do not receive a substituted inference method.

Covariance boundary diagnostics use feasible directions for the chosen family. A zero block cannot pass merely because derivatives in its standard-deviation coordinates vanish. Partial zero standard deviations also require a critical-cone curvature check, including equivalent AR1 correlation signs. If only separated AR1 coefficients have nonzero variances, a direct check of the first nonzero correlation prevents a zero derivative of a power of ρ from hiding a decreasing direction. At zero correlation the check also covers joint changes in correlation and zero standard deviations, using their leading covariance directions. These are local necessary optimality checks, not a guarantee of a global likelihood maximum. If a boundary check cannot certify the requested structure within its numerical work limits, the fit reports nonconvergence rather than declaring the boundary optimal. For these families, `object$engine$diagnostics$covariance.boundary.status` records whether the check was satisfied, a decreasing direction remains, or a numerical work limit prevented certification.

Computation

Grouping tuples use hash lookups while retaining first-observation level order. Sparse fixed designs use coefficient-space Woodbury products. Repeated design rows can also compress the pivoted QR for clearly full-rank designs; ambiguous rank decisions retain the observation-space QR. Fitted responses reuse the coefficient BLUPs. Full-rank categorical Type III tests can use repeated-row compression after checking that the treatment and sum designs span the same space; other cases retain covariance whitening.

Covariance optimization

By default, `optimizer="auto"` uses safeguarded average-information REML (AI-REML) for models with at most 200 free covariance parameters and only diagonal free covariance blocks. Other models, including ML fits, use L-BFGS-B. `optimizer="AI"` also permits unstructured and the structured random or residual covariance families described above in REML fits; models above the 200-parameter work limit use L-BFGS-B. `optimizer="L-BFGS-B"` selects L-BFGS-B directly.

Scalar sparse AI-REML models without supplied starts use a feasible residual-only initial covariance when common-scale profiling is enabled. Supplied parameters, explicit starts, fixed covariance blocks and zero iteration budgets retain their requested initialization.

Models with a structured residual family and no supplied covariance starts or fixed nonzero covariance normalize the common scale of the initial covariance shape before optimization. The multiplier is $y'Py/n$ for ML and $y'Py/(n-r)$ for REML. This is a starting-value calculation; `profile.scale` separately controls iterative scale profiling by AI. Explicit parameter evaluations and `maxit=0` retain their requested coordinates. These residual models allow up to twelve Newton refinement steps after the main optimizer, using the same objective-decrease and projected-score acceptance criteria as other models. Refinement does not relax the convergence tolerance.

AI uses analytic covariance-derivative actions and the fitted projection operator, without constructing an observation covariance derivative matrix for every parameter. Each update respects variance bounds and reduces its step until the likelihood improves. Singular information, failed step reduction or the `ai.maxit` limit transfers the accepted fit to L-BFGS-B with the remaining main iteration budget. Existing projected-score and positive semidefinite covariance boundary checks remain in force; a covariance rank restart uses L-BFGS-B. If the AI continuation fails or fails the score checks, one additional L-BFGS-B recovery starts from the original feasible parameters with the same `maxit` budget as a fresh L-BFGS-B fit. The better fit is retained. Reaching the main iteration limit alone does not trigger recovery. Finite-difference Newton polishing is retained only when the projected score still exceeds its tolerance. Polishing and boundary repairs are additional safeguards beyond the main iteration budget.

With `profile.scale=TRUE`, AI profiles a common covariance scale whenever all covariance blocks can be rescaled together and a free residual scale is available. A nonzero fixed covariance block disables profiling. The conditional REML scale optimum is $y'Py/(n-r)$ for the current covariance shape, where r is the estimable fixed-effect rank. Cached factors and operators are rescaled exactly, scalar sparse scores reuse their trace and quadratic parts, and the common-scale direction is removed from the AI update. `profile.scale=FALSE` disables this step. This does not assume centered responses, balanced groups or particular fitted means. Profiling does not remove an estimated parameter from AIC/BIC or change covariance-parameter uncertainty calculations. `maxit=0` evaluates the supplied starting coordinates without optimization or profiling.

Engine diagnostics add `optimizer`, `ai.iterations`, `ai.evaluations`, `ai.step.reductions`, `ai.fallback`, `ai.message`, `scale.profiled`, `scale.profile.updates` and `polish.evaluations`,

plus `ai.restarts` for the number of recovery attempts (zero or one). Evaluation counts record covariance evaluations; scale updates reuse factors and refresh scores separately. These fields identify fallback and polishing costs. The AI matrix guides optimization only; Satterthwaite and Kenward–Roger retain their existing information definitions.

Covariance computation

The sparse backend supports independent scalar and correlated random-coefficient terms, diagonal (possibly heterogeneous) residual covariance, and scalar random terms with relationship matrices. It forms penalized mixed-model equations in standardized random-effect space. Minimum-degree ordering and symbolic Cholesky analysis are reused across optimizer evaluations. Sparse numerical Cholesky and a selected inverse compute exact likelihoods and scores; no full observation inverse is stored.

For a coefficient covariance $G = LL'$, the transformed design is ZL. The symbolic pattern includes all possible nonzero coefficients over the fitted covariance family. Exactly zero or singular random covariance matrices therefore retain the same symbolic structure. Covariance derivatives are evaluated with grouped coefficient-space solves, without inverting G or dividing by its variances. Temporary derivative work is limited to batches of grouping levels. The relationship and REML projection corrections are retained exactly. This route supports the documented unstructured, diagonal, homogeneous diagonal, compound-symmetric and first-order autoregressive random-coefficient families, and fixed positive-semidefinite coefficient covariance matrices.

The sparse route requires the residual covariance to be structurally diagonal for every permitted covariance parameter value. A correlated coefficient block with its own supplied relationship matrix, or an observation covariance kernel, uses an exact covariance backend instead. For designs containing coefficient blocks, `engine$diagnostics$backend.reason` reports the sparse choice or fallback reason. Forced `backend="sparse"` rejects unsupported designs. Scalar relationship terms may coexist with independent correlated coefficient terms through the exact relationship correction.

With relationship-only random terms and one positive residual shape, the whitened relationship Gram matrix is cached for a single kernel as well as multiple kernels. A complete one-to-one relationship design whose absolute random-to-residual loading ratio is constant uses its existing relationship eigenbasis for diagonal variance updates. Every positive eigenvalue is retained; no ridge, rank approximation or numerical off-diagonal threshold is introduced. Other designs retain the cached full Gram calculation or the general solver. Diagnostics `relationship.gram.cached` and `relationship.spectral` identify these routes. For the diagonal route, `covariance.factorizations` is zero because variance evaluations use diagonal weights rather than a new covariance Cholesky factorization.

The block backend detects independent observation components from the covariance design. It factors those blocks separately and reuses factors only when the assembled matrices agree exactly after ordering. Ordinary correlated random-regression and multivariate bases remain supported by this route.

For ordinary multivariate and random-regression terms, the block route also searches design-based partitions with a small connecting random-coefficient system. It factors independent observation blocks and applies the exact Woodbury correction for those connecting effects. Every fitted covariance direction is retained, including exactly zero or singular random covariance matrices; no inverse of a random covariance matrix is required. This route is identified by `covariance.backend="block+random"`, with the connecting coefficient dimension in `structured.random.rank`. It currently excludes supplied relationship and observation kernels. The dense reference remains available through

backend="dense". Independent blocks that agree exactly reuse their factors, and unchanged base blocks and their design products are retained across variance evaluations. Covariance-score corrections are contracted in grouped coefficient space; corrected observation-block inverse matrices are not formed for those scores.

Trait-indicator bases with at most one nonzero per row use that exact support in covariance and score assembly. General bases retain the full calculation.

The sparse route combines multiple relationship terms in one exact Woodbury correction, with separate variance scales. For relationship-only models with one residual scale, it caches the unscaled Gram matrix and uses coefficient-space solves. Automatic selection keeps the dense reference when multiple relationship terms have at least as many levels as observations. The block route supports one scalar relationship term. Singular relationship matrices are supported without a ridge; every positive eigenmode is retained. Neither balance nor centered responses are required. Arbitrary kernels and multivariate relationship terms retain the dense reference route. Automatic selection uses matrix size and symbolic fill; it is a heuristic, not a timing guarantee.

Verified relationship-kernel product metadata are collected automatically in model$kernel.products and engine$diagnostics$kernel.products. The detector checks mapped products of other supplied relationship matrices, then complete ordered Kronecker products. detect.kernel.products=FALSE disables detection; kernel.product.tol sets the relative verification tolerance (default 1e-10). It preserves the supplied covariance and selects the computational routes described below when their design requirements are met. Further details appear in st.kernel.structure.

The default backend="auto" prefers the spectral product solver for eligible scalar relationship models with one positive diagonal residual shape. Its default spectral eigen driver prefers DSYEVD. Other model structures retain the automatic sparse, block or dense fallback. For a complete crossing with one observation per pair and constant absolute random-to-residual loading ratio, the solver uses the two factor eigensystems and tensor transformations. It does not form the full tensor eigenvector matrix. The tensor shortcut requires reconstruction agreement at floating-point roundoff; the detection tolerance does not relax this requirement.

For selected crosses, missing responses, replicated observations or nonconstant loadings, the solver diagonalizes the actual supplied residual-whitened SCA covariance on observed cells once. It then uses diagonal SCA/residual updates and an exact lower-dimensional correction for the remaining scalar random terms, including parental GCA. It never expands a subset to the full crossing and does not replace a near-product kernel by its reconstruction. Every positive relationship eigenmode is retained. Zero random variance components and singular parental covariances are supported without a ridge.

An exactly reconstructed product can also certify PSD through its two parental factors, avoiding a redundant large PSD decomposition. The factor spectra and every supplied covariance entry are checked before using this certificate. Near products retain the full PSD check.

These routes require fewer non-base random-effect levels than observed cells. The selected spectral route defaults to at most 32766 observed cells, the largest full DSYEVD order supported by the current 32-bit LAPACK workspace interface. control$kernel.max.obs can change this observation limit. Other designs retain the existing backends. kernel.solver=FALSE disables computational use while retaining detection metadata; backend="dense" selects the reference solver. Diagnostics kernel.product.solver report "complete", "selected" or "none"; kernel.product.correction.rank reports the retained correction dimension. The selected route still has a dense eigendecomposition at setup and is not a matrix-free solver.

control$eigen.solver="dsyevd" requires DSYEVD explicitly, "dsyevr" selects DSYEVR, and

"auto" specifies the default spectral policy: DSYEVD when its workspace fits the 32-bit interface, and DSYEVR otherwise. The workspace limit is checked before querying LAPACK. This fallback can be reached when an explicitly raised observation limit exceeds 32766; an explicit DSYEVD request is rejected at an unsupported order. Complete crossings retain the tensor shortcut using the factor eigensystems. The default mapped-product preference uses the user's configured thread count without changing it. It applies to eligible univariate scalar GCA/SCA models with three known random-effect covariance matrices, constant coefficient loadings, one independent residual, 6000–12000 observed responses and at least 16 configured threads. Complete products retain their spectral shortcut. The rule is based on measured solver comparisons; larger observation counts retain the prior eligibility rules. An explicit control$auto.dense.min.threads, or the option SelectionTools.auto.dense.min.threads, overrides this rule for incomplete products. A finite threshold selects dense when the configured thread count reaches that threshold; Inf disables the preference. Explicit backend choices take precedence.

No additional package or external solver is required. Dense products use R's BLAS and LAPACK, including the reference implementation. Fixed-effect estimation uses general GLS. Fixed-effect tests, predictions and their error covariances use the same fitted covariance operator. Sparse Type III tests use an exact rectangular whitening. Explicit joint prediction covariance, Kenward–Roger calculations and parametric bootstrap can still allocate dense intermediate matrices; sparse fitting does not remove those inference costs.

Scalar covariance blocks in the dense backend use local score accumulators by default. This reduces result-buffer writes while retaining the same likelihood and covariance model. control$dense.score.fast=FALSE selects the preceding generic score loop for comparison. Blocks with multiple coefficient dimensions retain that generic loop. The diagnostic dense.score.fast records whether the option is enabled.

Diagnostics report covariance.backend, covariance.blocks, covariance.max.block, covariance.factorizations and genomic.rank. For sparse fits, the first two size fields describe the independent residual rows; the factor count includes the sparse coefficient factor and any genomic correction. For block fits, it counts distinct observation-block factors plus any genomic correction; the structured route reports factors computed in the latest evaluation, excluding reused factors. Both exclude the fixed-effect factor. The genomic rank counts retained positive eigenmodes. sparse.equations and sparse.factor.nonzeros report the sparse coefficient dimension and Cholesky storage, including the diagonal. sparse.symbolic.analyses is one per sparse engine context, not per optimizer evaluation; separate inference reconstructions create new contexts. These diagnostics do not alter the statistical model.

Shared printing options

st.options() visibly returns a named list of the current printing options. Individual values are available through list extraction, for example st.options()$print.digits. A saved list can be supplied to st.options(saved) to restore the settings. Updates such as st.options(list(print.digits = 3L)) change only the named entries and return the complete updated list invisibly. Queries with no argument or NULL return the list visibly. The returned list is a snapshot; changes to that list do not change the settings.

The defaults are print.digits = 4L, print.char.perline = 75L, print.top = 10L, print.bottom = 5L, print.cov.blocks.top = 3L, print.cov.blocks.bottom = 1L, print.ems = FALSE, print.test.construction = FALSE, print.tests.for.bare.lsmeans = FALSE, and print.intercept.test = FALSE, print.LSD = TRUE, and print.HSD = TRUE. Digits range from 1 to 16; the line width ranges from 40 to 10000.

Leading and trailing row and covariance-block counts are nonnegative integers up to `.Machine$integer.max`, or `Inf` to display all rows or blocks in the corresponding section. The logical switches accept a single nonmissing logical value. Unknown, duplicate, missing and unnamed option names are errors. Invalid updates leave every setting unchanged.

SelectionTools printers use these session settings when their corresponding arguments are omitted or `NULL`. Explicit printing arguments take precedence: `digits`, `char.per.line`, `top` and `bottom` override the four formatting options. For model output, `ems` in `...` overrides expected mean square printing for method-of-moments fits. `test.construction` in `...` overrides test construction printing for both likelihood and method-of-moments fits. The ANOVA type appears where applicable when test construction is displayed. For least-squares means output, `tests` overrides `print.tests.for.bare.lsmeans`; contrasts retain their tests by default. The switch `print.intercept.test` controls stored intercept tests in ANOVA and fixed-effect test tables; `intercept.test` in `...` overrides it for an individual print. Regression coefficient reports retain the intercept estimate, standard error and test independently. A fixed-effect test section with no displayed rows is omitted. For an intercept-only fixed model, it appears when the intercept test is displayed. The switch does not add an intercept test to moment ANOVA source tables that contain no such test. Settings affect display only and do not change stored values, fitting, inference or R's global options. Each newly loaded package DLL starts with the defaults. The switches `print.LSD` and `print.HSD` independently control pairwise difference thresholds beneath least-squares means without displayed tests. The thresholds remain stored when either switch is false.

Method-of-moments output

Expected mean squares retain full factor names. Long expressions fill the available line width and continue between variance components. Across designs, the ANOVA table contains the numerical results; optional test-construction blocks identify the numerator and denominator mean squares. Expected mean squares and test-construction blocks are hidden by default and are available through the corresponding shared options or explicit `print` arguments. ANOVA and extracted F-test tables omit the Kind and error-source columns. The optional EMS and test-construction blocks follow variance or covariance component estimates when these are present. Trait labels distinguish results for multiple responses, and separate residual blocks distinguish multiple error strata. Test-construction blocks identify exact or approximate tests and sources without a requested or available test. These display choices leave stored ANOVA tables, tests and estimates unchanged.

Likelihood test construction

The optional test-construction block describes the displayed tests after the variance or covariance component estimates. Exact F tests identify the denominator mean square from the fitted design when that source was retained. Wald tests identify the denominator DF method, and likelihood ratio tests identify the comparison with a reduced model and the reference distribution. The block uses stored results and requires no additional model fitting.

Printing extracted results

The ordinary `print` generic uses the same compiled `digits`, `top`, `bottom` and `char.per.line` controls for models and extracted results. For example, `print(vcov(m), digits=3, top=5, bottom=2, char.per.line=60)` formats the coefficient covariance matrix. There is no separate printing function.

Numeric results from coef, vcov, fitted, residuals, model.matrix, predict, confint, logLik, AIC, BIC, nobs and df.residual carry the additional S3 class st.lmm.result. AIC and BIC return data frames with one row per model, a DF column for the parameter count and a column for the criterion. Formula results retain their formula class and environment. Vectors remain numeric or integer vectors; matrices retain their dimensions, labels and matrix inheritance. Numbers retain full precision. The same display class is used for coefficient and covariance accessors of means and prediction objects. Existing specialized printers handle their summaries and test tables.

nobs counts the observed response cells, while df.residual subtracts the rank of the full fixed design from this count. st.lmm.components(object, "observations") also returns the retained row count, observed count by trait, original row positions and observed cell indices in column-major response order. For moments fits, DF.error.stratum records the error-stratum DF by trait. These DF remain the basis for the applicable inference. The observation record and the model statistics report fitted.type, which identifies the target used for the default response residual and its RMSE. RMSE uses residuals on the original response scale. A weighted moments fit also reports RMSE.weighted, based on its precision-weighted residuals. The fit matrix in the prediction list has the same layout as fitted values.

The covariance axes of an intercept-only multivariate block are named by trait in model order. Covariance blocks with multiple coefficients within a trait retain their coefficient names. The ordinary residual block is named units. Named covariance inputs use the fitted axis names or the corresponding coefficient names.

New fits also provide this print method for m$engine$sigma, m$engine$u and m$engine$diagnostics, including their extracted components. It also formats the table returned by anova(m, effects="both"). For these lists, top and bottom limit displayed components and the rows within each component; either limit may be Inf. The model summary retains its separate three-covariance-block limit.

Printing returns the original object invisibly. It changes no values, labels, random-number state or global print options. Arithmetic, mathematical functions, summary operations and ordinary conversions use the original R classes. Subsetting and transposition retain the result printer. Standard conversions such as as.numeric, as.vector, as.matrix and as.data.frame provide ordinary R values when needed by other code.

Printed summary

The ordinary full-model summary begins with the fixed, random and residual model formulas, followed by convergence and tests. Fitting controls remain in object$call. When subset comparison was requested, the default summary returns that comparison instead; summary(object, selection=FALSE) gives this full-model summary.

When the fixed formula contains numeric predictors, a fixed-effect regression estimate table follows the random-effect tests and precedes the variance components. It includes the intercept, if fitted, and terms involving numeric predictors, with estimates, standard errors, denominator DF, t ratios and unadjusted two-sided p-values. Purely categorical coefficients are omitted; st.lsm provides adjusted factor means. Numeric-by-factor terms are retained: Diet:Time can give level-specific slopes, whereas Diet*Time contains a reference slope and slope differences with treatment contrasts. The labels follow the fitted design coding. Reference levels are stated when the fitted contrast matrix uses a single reference level. Transformations and matrix-valued numeric bases are included. Multivariate tables are separated by response. Likelihood fits use the fitted Satterthwaite, Kenward–Roger or asymptotic method. Ordinary fixed-only moment fits use exact t tests. Unavailable and non-estimable results are labelled without changing the inference method. New fits retain

this numeric table in `object$regression`; it is NULL for models without numeric fixed terms. Older saved fits construct the table privately when printed, without changing the saved object.

For method-of-moments analyses with a single error term, the ANOVA table includes a final `Residual` row with its degrees of freedom, sum of squares and mean square. The residual row has no F-test or p-value. This layout also applies to models with random effects, including nested villages within regions. Analyses with multiple residual error terms retain their separate residual table. The stored tests and fitted estimates are unchanged by this display.

A variance-component table is always printed, including `Residual` (the residual variance) when no random term is specified. Heterogeneous residual blocks are shown separately. For multivariate responses and random regressions, row and column labels distinguish variances on the diagonal from covariances off the diagonal. The session settings `print.cov.blocks.top` and `print.cov.blocks.bottom` select the leading and trailing covariance blocks. An omission count separates the displayed groups when applicable. A blank line follows each displayed block. A full labelled covariance matrix is printed when its rows fit `char.per.line` at the requested precision. Wider matrices use a lower-triangle table. Scalar variances use a separate Component/Variance table with its own row limits. All estimates are retained in `object$engine$sigma`. Random-regression coefficients are retained but not printed. The named list `object$engine$u` holds matrices with grouping levels as rows and random basis coefficients as columns. For example, `object$engine$u$Chick` contains the chick-specific random intercepts and slopes for `~Time | Chick`. These are predicted random deviations (BLUPs). Their sum with the corresponding fixed regression contribution gives an individual regression. A random-slope-only model has no random-intercept column.

Fits criteria and model statistics are combined at the end in two side-by-side columns of labelled values. Multivariate summaries show the overall fit criteria once and identify each trait's statistics separately. Numbers use fixed decimal notation; small p-values and large F-statistics are displayed as bounds at the stated thresholds. Random-test sections are omitted when tests were not requested or there are no random terms; variance estimates are always shown. Printing does not modify the underlying estimates or tests. Printed headings use `DF1` and `DF2` for numerator and denominator degrees of freedom, `DF` for a single degrees-of-freedom column, and `F`, `t`, and `p` for test statistics and probabilities. Wrapped model formulas use hanging indentation; formulas in a random-term list are shown on separate lines. The data frames returned by `anova` use the same compiled printer while retaining their stored column names.

Designed experiments and denominator DF

For balanced split plots, Satterthwaite and Kenward–Roger reproduce the exact error-stratum DF when the corresponding variance components are estimable. In the balanced split-plot datasets `v-lmm-alfalfa` and `v-lmm-oats`, the variety effect has 10 denominator DF, and subplot effects and their interactions have 45. An explicitly requested `ddf="asymptotic"` instead uses an infinite denominator DF.

SAS containment uses rank contributions of syntactically containing random terms, or $n-\mathrm{rank}[X\ Z]$ if none contains the fixed term. It is not currently an `st.lmm` option. In incomplete-block and lattice mixed models, Satterthwaite and Kenward–Roger can have fractional DF that differ from containment or a least-squares analysis treating blocks as fixed. Comparable results require the same model and DF method.

External inference interfaces

When users install **emmeans** or **multcomp**, their inference functions can consume supported `st.lmm` objects directly. SelectionTools does not load these packages and does not require them for installation or fitting. Methods become available when the corresponding external namespace is loaded. All SelectionTools-side calculations use compiled C routines with thin R entry points. No interface refits the model or calls an optimizer.

For univariate likelihood fits, `emmeans::emmeans(object, ~ factor)` uses retained predictor data, the compiled fixed-effect design, covariance and estimability information. The optional `ddf` argument selects `"Satterthwaite"`, `"Kenward-Roger"`, or `"asymptotic"`; the fitted object's method is the default. Degrees of freedom are calculated for each requested linear combination. Kenward–Roger requires REML and uses the matching adjusted covariance. Rank-deficient designs retain a non-estimability basis. Unavailable finite-sample inference is not replaced by asymptotic inference.

`multcomp::glht(object, linfct = K)` and supported `multcomp::mcp` comparisons use fixed coefficients and their matching covariance. By default `glht` uses its normal approximation. A user may supply one finite nonnegative `df` at the **multcomp** interface; positive values request its common-DF t inference, and zero requests the normal approximation. Contrast-specific Satterthwaite degrees of freedom are not collapsed into a common value.

The first implementation excludes multivariate fits, moments fits, selected-model collections, emmeans data/covariance overrides, argument-supplied offsets, and glht coefficient/covariance overrides. Formula offsets are supported by emmeans. The glht interface rejects rank-deficient fits; the estimability-aware emmeans interface supports those fits. Interaction comparisons follow the external package's own contrast and averaging conventions. Automatic trends, transformations, and advanced external options require validation for the model concerned.

See Also

`st.lmm.profile`, `st.predict`, `st.lsm`, `st.get.A`

Examples

```
saved.options <- st.options()
st.options(list(print.digits = 3L, print.ems = TRUE))
st.options(saved.options)
```

---

st.lmm.allfit *Compares Gaussian mixed-model optimizers and starting values*

Description

Refits a compiled Gaussian model with the available optimizers and specified starting values. Every attempt uses the same observations, fixed and random designs, covariance families, likelihood definition and parameter bounds. Successful, nonconverged and failed attempts are retained separately.

Usage

```
st.lmm.allfit(object, optimizers = NULL,
  starts = c("fitted", "default"), control = list(), keep.fits = TRUE)

## S3 method for class 'st.lmm.allfit'
summary(object, ...)
## S3 method for class 'st.lmm.allfit'
print(x, ...)
```

Arguments

`object`	An ML or REML `st.lmm` fit; for `summary`, the result of `st.lmm.allfit`. The original fit need not have converged, but each selected starting vector must define a finite Gaussian likelihood.
`optimizers`	Unique selections from `"AI"` and `"L-BFGS-B"`. The default compares both for REML and uses `"L-BFGS-B"` for ML. Models with explicit covariance constraints or a US trait-by-occasion residual product use only `"L-BFGS-B"`; an explicit AI request for such a model is rejected. Average information is available for REML. Its safeguarded implementation may continue with L-BFGS-B; the requested and actual optimizer and full fallback diagnostics are recorded.
`starts`	Unique selections of `"fitted"` and `"default"`, or a uniquely named list of numerical coordinate vectors in the order returned by `st.lmm.objective`. Named vectors must match that exact coordinate order. The default starting vector is constructed from the retained model with estimated covariance starts cleared. Every selected optimizer receives exactly the same vector for a given start.
`control`	Common fitting controls. This list must not set `optimizer`, `parameters` or `evaluate.only`; the first two are determined by the comparison arguments.
`keep.fits`	Logical. Full fitted objects are retained in addition to estimates and diagnostics when TRUE.
`x`	A `st.lmm.allfit` object.
`...`	Display controls for the result-table printer. Omitted controls use `st.options`.

Details

Repeated fits run in the current R process. Numerical kernels use the existing OpenMP and BLAS thread controls; `st.set.num.threads` describes these controls. Intermediate state stays in C-owned memory.

This function compares the optimizers implemented by `st.lmm`; it does not call external optimization packages. Repeated fits reuse the retained model and do not reread mutable data or a marker-data registry. Subset selection and random-effect tests are disabled in these repeated fits, and fixed-effect inference uses the asymptotic setting so that the comparison measures fitting rather than additional inference procedures. The supplied model is not changed or automatically replaced.

Explicit fixed/shared covariance constraints are retained for all attempts, including compact results with `keep.fits=FALSE`. Each start uses the same physical parameter declarations. Compact estimates retain the constraint inventory and independent covariance-parameter count.

The summary uses the converged attempt with the highest log likelihood as its reference. It reports the initial and final deviance, deviance difference, projected gradient, covariance-cone score, rank, differences in fixed estimates, root mean squared difference of fitted values, and absolute and relative Frobenius differences of physical covariance-block matrices. Fixed-coefficient differences are omitted if the retained estimable coefficient basis changes. If no attempt converged, reference comparisons are missing. A nonconverged attempt with a better objective can have a negative deviance difference and must be investigated rather than treated as a valid reference fit.

Warnings, error messages, timings, coordinates, bounds, estimates and execution identities remain in individual task records.

Value

A `st.lmm.allfit` object containing the original fit, task inventory, initial objective records, all attempt records, controls, execution metadata. `summary` returns a data frame with one row per attempt. Its `reference.id` attribute identifies the successful reference attempt, or is missing when none succeeded.

See Also

`st.lmm`, `st.lmm.refit`, `st.lmm.objective`, `st.lmm.bootstrap`, `st.lmm.profile`

---

st.lmm.bootstrap *Bootstrap Statistics from Gaussian Linear Mixed Models*

Description

Generates responses from an ML or REML fit, refits its retained model design, and evaluates a named numeric statistic. Replicate records include the generated response, random-number stream, convergence diagnostics, warnings and failures.

Usage

```
st.lmm.bootstrap(object, statistic = stats::coef, nsim = 999L,
  seed = NULL, type = c("parametric", "residual"), re.form = NA,
  statistic.args = list(), responses = NULL, control = list(), series = NULL)

## S3 method for class 'st.lmm.bootstrap'
summary(object, ...)
## S3 method for class 'st.lmm.bootstrap'
print(x, ...)
## S3 method for class 'st.lmm.bootstrap'
confint(object, parm, level = .95,
  type = c("percentile", "basic", "normal"), allow.failed = FALSE, ...)
```

Arguments

object	For bootstrap generation, a converged st.lmm ML or REML fit. For summary and intervals, an st.lmm.bootstrap result.
statistic	A function whose first argument is a refitted model. It must return a nonempty finite numeric vector with unique, nonempty names in the same order for every replicate. The default returns fixed-effect coefficients. A rank-deficient model requires a statistic selecting estimable coefficients or combinations instead of returning aliased entries.
nsim	Positive integer number of replicates.
seed	An integer master seed, or NULL to draw one from R's current random-number state. A supplied seed restores the caller's previous RNG kind and state, including the absence of .Random.seed. With NULL, drawing the recorded master seed advances the caller's state.
type	For generation, "parametric" samples Gaussian random effects and residuals; "residual" resamples centered and scaled conditional residuals in the limited setting described below. For confint, the interval construction.
re.form	Random coefficients held at their fitted values during a parametric bootstrap; simulate.st.lmm describes the generation step. The default NA regenerates all random effects. NULL holds all of them. Selectors may hold complete blocks or fitted coefficients within a block. Residual resampling always holds every fitted random coefficient and requires the default re.form=NA; it has no selectable conditioning argument.
statistic.args	A list of additional named arguments passed to statistic after the fitted model.
responses	Optional externally generated responses. A list of response matrices has one matrix per simulation, with one row per retained observation and one column per trait. A univariate model also accepts a list of vectors or a matrix with one simulation per column. The number of responses must equal nsim. Each response uses the retained layout of model.response(model.frame(object)). Original missing response cells remain missing; an observed cell cannot become nonfinite.
control	Named fitting controls for repeated fits. The ordinary route passes them to st.lmm.refit. The joint series route applies the same controls to the compiled first- and second-stage refits; physical control$parameters overrides are not supported there.
series	NULL for the ordinary bootstrap, which holds any inherited first-stage covariance fixed. Refitting both stages uses the original earlier fits supplied as list("env", fit1, fit2, ...) in their original order. The object must be a series fit with its complete original target cells. Further details appear in 'Joint bootstrap of a series'.
x	A bootstrap result to print.
parm	Statistic names or distinct positive indices to include in an interval table. Omission includes all statistics.
level	Confidence level strictly between zero and one.

`allow.failed` By default, intervals are refused if any replicate failed. Explicit `TRUE` permits intervals from successful replicates and produces a warning and an attribute stating that inference is conditional on fitting success. At least two successes are required.

`...` Printing arguments, or additional generic arguments. Unsupported interval arguments are rejected.

Details

Repeated fits run in the current R process. Numerical kernels use the existing OpenMP and BLAS thread controls; `st.set.num.threads` describes these controls. R callbacks execute on the main R thread. Intermediate state stays in memory.

Bootstrap stream scoping requires a generator whose complete state can be saved. Box-Muller and user-supplied generators are rejected before any RNG state is changed. The default Inversion normal generator is supported. The simulation covariance factorization is prepared once per run and reused for every parametric response. Responses are generated before fitting, using the stream assigned to each replicate. The covariance factorization is reused across responses.

The parametric method uses the fitted physical covariance matrices and the observed fixed-effect estimate. It includes the fitted offsets and observation precision weights. By default, random coefficients and residual errors are regenerated independently. All model matrices, grouping indices, factor coding, transformation calibration, known covariance matrices and missing-response positions are retained during refitting. A subset-selection search or an internal random-effect bootstrap test is not repeated.

When selected random coefficients are held, they are treated as fixed plug-in constants. Unselected coefficients are generated from their marginal fitted covariance, with both covariance axes limited to those coefficients. This does not draw their conditional distribution given the held coefficients. It does not propagate uncertainty in the held fitted values. The resulting conditional target determines the interpretation of a statistic.

Residual resampling requires a univariate model with one homogeneous, positive, independent residual variance. Heterogeneous or correlated residual errors and multivariate responses are rejected for this method. Observed conditional residuals are centered and scaled so that their empirical second moment equals the fitted observation-level residual variance. They are sampled independently with replacement and added to the full fitted conditional mean. Constant precision weights are allowed if the resulting residual covariance is homogeneous. This is an approximation: centering and rescaling do not remove the shrinkage and dependence induced by fitting. In particular, holding random effects fixed omits their between-sample variation and can give inadequate coverage for population-level parameters. The parametric method is the primary Gaussian mixed-model bootstrap.

With `series=NULL`, the statistic is an R callback evaluated on the main thread. It should use the fitted model and explicit arguments; deliberate random draws use the assigned replicate stream. Reproducibility requires that the callback does not depend on mutable external state or external side effects.

The original statistic is evaluated in a separately recorded substream. Each replicate ID has a separate `"L'Ecuyer-CMRG"` stream. Responses are generated in the parent process before the repeated fits; the resulting post-generation stream is used for a statistic that draws random numbers. Stream assignment is determined by replicate ID. Floating-point fitting results can differ slightly across numerical configurations.

Joint bootstrap of a series. Supplying `series` uses a parametric bootstrap which regenerates plot responses, refits each earlier model, recomputes its adjusted means and full covariance, and refits the second-stage model with that new covariance. Numerical arrays, simulation roots and intermediate fitted contexts are compiled C memory. No intermediate R fitted models or R numerical workspaces are built. The earlier input fits and second-stage object are not changed.

The initial joint scope is univariate converged Gaussian ML/REML models, independent first-stage errors, `type="parametric"`, `re.form=NA`, `statistic=stats::coef`, and no `statistic.args` or externally supplied `responses`. Earlier provenance/order, targets, means and within-fit covariance must match the series object. Second-stage subsets or reordered rows, multivariate joint generation, nonzero cross-fit error covariance, residual resampling, held random effects and custom statistics are rejected. Explicit reference grids, named row weights, offsets, missing plot responses and supported physical covariance constraints are retained. The target covariance of each earlier fit must be positive definite; no ridge is added.

The matrix L_e defines the earlier fixed-effect targets, $C_e = L_e\widehat{\mathrm{Var}}(\widehat{\beta}_e)L_e^T$, and $\widehat{m}_e$ includes their reference offsets. The second-stage target signal θ^* is drawn from its fitted fixed mean, random effects and any additional estimated residual components, excluding the known inherited covariance. For each earlier trial, the lifting matrix is

$$A_e = X_e\widehat{\mathrm{Var}}(\widehat{\beta}_e)L_e^T C_e^{-1}$$

and the plot responses are drawn with mean

$$X_e\widehat{\beta}_e + o_e + A_e(\theta_e^* - \widehat{m}_e)$$

and its original fitted plot-level marginal covariance. Thus target variation from the second-stage model and first-stage plot sampling error are generated separately; inherited sampling error is generated once. This covariance-based lifting fixes a reproducible choice of the earlier nuisance mean when targets use different covariate or nuisance-factor references. Earlier designs, contrasts, transformations, masks and reference definitions remain fixed. Every replicate re-estimates the free covariance parameters at both stages. This is plug-in parametric generation; it is not a coverage guarantee or a replacement for fitting a joint plot-level likelihood.

The result retains the ordinary bootstrap class and has attribute `bootstrap.stage="joint series"`. The original earlier fits are supplied explicitly when calling the function; they are not retained as backend working objects. Each record's `response` is a named list of generated earlier plot-response vectors; its `latent.targets` attribute records the generated second-stage target signal. `record$fit$stage1` records each compiled refit's status, covariance components, adjusted means and recomputed target covariance; `record$fit$stage2` records second-stage diagnostics and covariance components. Failed rows remain `NA`, with available stage diagnostics, and are not replaced. Existing interval failure rules continue to apply. Summary and interval tables retain the bootstrap-stage attribute.

The quantities t_0, $\bar{t}^*$, s^*, and q_a denote the original statistic, its bootstrap mean, its sample standard deviation, and its bootstrap quantile of order a, respectively. Percentile intervals use the empirical lower and upper quantiles, with linear interpolation equivalent to R's `quantile(type=7)`. Basic intervals reflect these limits around t_0. Normal intervals use $2t_0 - \bar{t}^* \pm z_{(1+level)/2}s^*$, including the bootstrap bias correction. These are approximate intervals; nominal coverage is not guaranteed in small samples or at variance boundaries. BCa and studentized bootstrap intervals are not implemented by this interface.

Value

An st.lmm.bootstrap object contains t0, the original named statistic; t, the replicate-by-statistic matrix with failed rows retained as NA; records, the full per-ID responses, streams, statuses, warnings, errors, diagnostics and execution information; and seed, the original model and statistic call. Original-statistic warnings are stored separately.

summary returns the original estimate, bootstrap mean, bias, sample standard deviation, and counts of successes and failures. confint adds lower and upper limits, retaining statistic names and requested order. An interval table records its type, level and whether it conditions on successful fitting. No failed replicate is silently replaced.

See Also

st.lmm, st.lmm.refit, st.lmm.simulate, st.lmm.profile

---

st.lmm.by *Fits Independent Gaussian Models to Groups of Observations*

Description

Observations are split by a grouping variable, and the same model specification is fitted separately in each group. The result retains each fitted model, its original row mapping, warnings and fitting status. Fixed-only univariate regressions can optionally use a common pooled residual variance for coefficient inference.

Usage

```
st.lmm.by(fixed, group, data, random = NULL, cov = ~units,
          method = "REML", ddf = "Satterthwaite", type = 3L,
          random.test = "none", control = list(), ucov = NULL,
          weights = NULL, contrasts = NULL, subset = NULL,
          na.action = NULL, offset = NULL, pool = FALSE)

## S3 method for class 'st.lmm.by'
summary(object, pool = NULL, level = .95, ...)
## S3 method for class 'st.lmm.by'
coef(object, ...)
## S3 method for class 'st.lmm.by'
confint(object, parm, level = .95, pool = NULL, ...)
## S3 method for class 'st.lmm.by'
print(x, digits = NULL, top = NULL, bottom = NULL,
                        char.per.line = NULL, ...)
## S3 method for class 'summary.st.lmm.by'
print(x, digits = NULL, top = NULL,
                                  bottom = NULL, char.per.line = NULL, ...)
```

Arguments

fixed	A two-sided fixed-effect formula, as in st.lmm.
group	A single data-column name, a formula of the form ~column, or an atomic grouping vector with one value per original data row. A column symbol can be used directly. Groups are ordered by their first appearance after subsetting. Selected group values must not be missing or nonfinite.
data	An R data frame or other data representation accepted by st.lmm, including a bitpacked SelectionTools marker data set or a SelectionTools hybrid data set.
random, cov, ddf, type, random.test, control, ucov, contrasts	Model specification and fitting arguments passed to each independent fit. Further details appear in st.lmm.
method	"ML" or "REML". Method-of-moments fitting is not supported by this grouped interface.
weights	Optional numeric observation precision weights, evaluated on the original data and then sliced for each group. The validation and zero-weight rules of st.lmm apply within each group.
subset	Optional logical or integer row selection, evaluated on the original data before grouping. Repeated integer indices are retained.
na.action	A standard missing-value action, applied separately within each group. NULL retains the compatibility behavior of st.lmm.
offset	An optional known numeric offset, evaluated on the original data before grouping. A scalar is recycled. It is added to any offsets specified in the fixed-effect formula.
pool	Logical for st.lmm.by. TRUE requests coefficient inference using a common residual variance across groups. This requires fixed-only univariate models with cov = ~units. In summary and confint, NULL uses the choice made when fitting; TRUE or FALSE requests pooled or separate-model inference.
object, x	An st.lmm.by object, or its summary for the corresponding print method.
parm	Optional coefficient names or indices into the columns of coef(object). Omission selects all coefficients.
level	The confidence level, strictly between zero and one.
digits, top, bottom, char.per.line	Per-call display controls, as in st.lmm.
...	Additional generic-method arguments; currently unused.

Details

Repeated fits run in the current R process. Numerical kernels use the existing OpenMP and BLAS thread controls; st.set.num.threads describes these controls. R callbacks execute on the main R thread. Intermediate state stays in memory.

The models are independent fits. In particular, a random-effect level with the same label in two groups does not share an estimated random coefficient between those groups. Fixed-effect estimates and covariance parameters are estimated separately. Shared effects across groups are represented by the grouping structure in a single st.lmm model.

Grouping, weights, the argument offset and the subset are evaluated on the original data. The subset is applied before splitting. Each subgroup is then treated as a fresh data set: transformations such as `scale(x)` or `poly(x, 2)` are learned from that subgroup, before its missing-value action. Declared factor levels and contrast attributes are retained as in ordinary data-frame subsetting. Aliased coefficients remain explicitly marked as not estimable. The prepared subgroup designs, evaluated inputs, contrasts and referenced formula bindings are retained, so saved tasks do not depend on later changes to the original data or caller bindings.

Marker or hybrid data and any requested relationship matrices are resolved before fitting the groups. The retained subgroup requests can therefore be evaluated after the corresponding in-memory data set has been removed. The original marker or hybrid representation remains unchanged.

An error in an individual group's preparation or fit is retained in that group's result. A fit that does not converge is retained with a separate status. Groups are never silently dropped, and coefficient inference is unavailable for unsuccessful fits. The result's `complete` field is true only when every group fit converged. Each ordinary fitted model uses its local data-row mapping; the grouped result additionally records original data-row indices and positions in the selected layout, including duplicates.

Pooling assumes a common residual variance after accounting for any known precision weights. It is available only for fixed-only univariate models with one independent residual-scale component. Here e_g is the residual vector in group g, W_g is its diagonal precision matrix, n_g is its number of observations and r_g is its fixed-design rank. The pooled variance is

$$\widehat{\sigma}^2 = \frac{\sum_g e_g^{\mathsf{T}} W_g e_g}{\sum_g (n_g - r_g)}.$$

The coefficient covariance in each group is this variance times the inverse weighted fixed-design information on its estimable columns. Confidence intervals and tests use the Student distribution with $\sum_g (n_g - r_g)$ degrees of freedom. No pooled inference is produced unless all required fits succeed and the pooled residual degrees of freedom are positive.

`summary` adds a coefficient table with estimates, standard errors, degrees of freedom, test statistics, probabilities, confidence limits and inference statuses. It retains the group inventory and reports whether pooled inference was requested. `coef` returns a matrix with one row per group and the union of coefficient names as columns; unavailable or aliased coefficients are `NA`. `confint` returns the coefficient table containing only the selected coefficients, retaining explicit rows for failed groups. The interval level and inference interpretation are stored as attributes. These methods do not modify the fitted object.

See Also

`st.lmm`, `st.lmm.refit`, `st.lmm.components`

`st.lmm.compare` *Likelihood Comparisons and Fixed-Term Deletion for Mixed Models*

Description

Compares fitted univariate Gaussian mixed models using likelihoods and information criteria, or refits models after deleting eligible fixed terms. Likelihood-ratio inference is supplied only when nesting has been established.

Usage

```
## S3 method for class 'st.lmm'
drop1(object, scope, test = "none", k = 2,
                      trace = FALSE, ..., refit = TRUE)
```

Arguments

object	A converged univariate st.lmm model fitted by ML or REML.
scope	Optional character vector of fixed-term names or a one-sided formula identifying those terms. By default, all terms eligible under the marginality hierarchy are considered. An interaction prevents deletion of its constituent lower-order terms. The intercept is not a deletion term.
test	"none" reports information criteria; "Chisq" also reports likelihood-ratio statistics and approximate chi-squared probabilities. The multi-model anova interface defaults to "Chisq"; drop1 defaults to "none".
k	Finite nonnegative information-criterion penalty per estimated parameter. The default 2 gives AIC; log(nobs(object)) gives BIC.
trace	Logical; TRUE prints the term being deleted.
refit	Logical. For multi-model anova, TRUE refits every REML model by ML. FALSE keeps the fitted method and permits REML comparison only with identical fixed design matrices. Fixed-term deletion requires ML, so a REML object is refitted when TRUE and rejected when FALSE.
...	Reserved for drop1. For multi-model anova, additional fitted models are unnamed arguments.

Details

anova(m1, m2, ...) compares ML or REML st.lmm objects. The effects="fixed" and effects="random" selections return the corresponding tests cached during fitting. Both effects="both" and effects="all" return the combined table. Moments-model comparisons retain their separate exact-regression behavior.

All compared models must retain identical response values, observation identities and order, precision weights, and offsets. This includes the layout of any retained missing-response prediction rows. Models fitted to different observations cannot be compared through this interface. Every supplied fit must have a finite likelihood and report convergence.

The default ML refitting makes comparisons of different fixed effects meaningful. Refitting is reported in the table note and its ML.refitted attribute. The supplied objects are not changed. With refit=FALSE, ML and REML cannot be mixed. REML comparisons also require the same numeric fixed design matrix: using different contrast bases can change the additive constant in a REML likelihood even when the fixed model spaces coincide.

Models are ordered by their number of free parameters. This count is the estimable fixed-effect rank plus the nominal number of free covariance parameters. A fitted variance of zero does not remove its parameter from this count. AIC and BIC use this count and the observed response count. The input.order attribute maps table rows to the supplied order. The optional model.names argument to anova supplies unique nonempty labels; otherwise concise, distinct model expressions are used with numbered labels as a fallback.

For consecutive models with a positive parameter-count difference, nesting is checked using the fixed design spaces and the random covariance families. Supported random comparisons include removal of free random blocks, scalar random coefficients within a larger unstructured block, and independent coefficients within a corresponding unstructured block. The homogeneous-diagonal, compound-symmetry and AR(1) families are compared only when their covariance constraints establish inclusion. The residual covariance families and physical bases must agree. For unproved nesting, the table retains information criteria and reports unavailable LRT inference with an explanation. Equal parameter counts also give no LRT probability. No test is inferred from parameter counts alone.

Covariance-parameter LRTs use an approximate chi-squared distribution. Variance parameters on the boundary of their allowable range can require a different calibration; such comparisons have an explicit table note. For supported whole-random-term tests, `st.lmm(random.test="bootstrap")` provides the existing parametric-bootstrap alternative. A fixed-term likelihood-ratio test differs from a cached Wald F test and does not use Satterthwaite or Kenward–Roger denominator degrees of freedom.

`drop1` removes one complete eligible formula term per fit. A factor term can therefore remove several estimable coefficients. The same observations, offsets, weights and covariance specification must remain in use. Dropping an aliased term can leave the estimable rank unchanged; the table then gives no invented degrees of freedom or probability. An unsuccessful reduced fit is retained in the table with an explanatory `status` and unavailable inference. An incompatible observation layout is an error. Models are reconstructed from retained input and formula metadata rather than a mutable caller data binding.

Value

A data frame using the package's table printer. Multi-model `anova` reports `model`, `npar`, `AIC`, `BIC`, `logLik`, `deviance`, `Chisq`, `DF`, `p` and `comparison`. The `comparison` column identifies the reference row, nested tests, or unavailable inference.

`drop1` reports `term`, `DF`, `AIC`, `LRT`, `p` and `status`; the first row is `<none>`, the full model. Its `AIC` column uses the supplied penalty `k`. The printer labels the column AIC for `k=2`, BIC for `k=log(nobs(object))`, and Criterion for other penalties. The `method` and `ML.refitted` attributes record the estimation convention.

See Also

`st.lmm`, `st.lmm.model`, `anova`, `drop1`

---

st.lmm.components — *Components and Random-Coefficient Covariance Diagnostics of a Fitted Model*

Description

Extracts coefficients, covariance components, model construction and estimability information from a Gaussian linear mixed model. The related functions diagnose the numerical rank of each random-coefficient covariance block and return its principal components. Construction and calculations are performed in C.

Usage

```
st.lmm.components(object, name = NULL, traits = NULL, block = NULL,
  term = NULL)
st.lmm.singular(object, tol = 1e-4)
st.lmm.pca(object, tol = 1e-7)
```

Arguments

object	A univariate or multivariate st.lmm fit. st.lmm.components accepts ML, REML and moments fits. st.lmm.singular and st.lmm.pca accept ML and REML fits.
name	NULL for the standard component list, or a character vector of component names listed below. A single name returns the component itself. Several names return a named list in the requested order. Names must be nonempty, distinct and nonmissing. Matching is exact.
traits	NULL or a character vector of distinct fitted response trait names. The variance, covariance and correlation components support this selection. An explicit selection uses the supplied trait order; NULL uses fitted order.
block	NULL for all covariance blocks, or one exact fitted covariance block name. The variance, covariance and correlation components support this selection. The result for one selected covariance or correlation block is its matrix. The variance matrix retains one row for the block.
term	One exact, nonmissing and nonempty hypothesis term name for sscp.H. This argument is required for sscp.H, including requests with several component names. It is NULL for other requests.
tol	A finite number strictly between zero and one. A principal component is counted in the numerical rank when its standard deviation, divided by the square root of the reported reference variance, is strictly greater than tol. Further details appear in Details for the reference scale.

Details

Components: The standard result contains the following entries.

fixed Named fixed-coefficient estimates in the full original model-matrix order. Aliased columns retain their NA placeholders. This is also the default coefficient content of coef(object).

random A named list of random-coefficient predictions. An ordinary random block is a matrix with group levels in rows and coefficient names in columns. A direct covariance-kernel block has a single column with one prediction per response cell. Moments fits and fits without random terms return an empty list.

vcov.fixed The model-based fixed-coefficient covariance matrix at the fitted covariance parameters. It is unadjusted even when the fit requested Kenward–Roger inference. This extractor does not change the adjustment policy used by vcov(object).

covariance A list with random and residual entries, each containing the fitted coefficient covariance matrices in their original coefficient units. The axes of an intercept-only multivariate block identify the traits. In a likelihood fit, known covariance between grouping levels contributes to the full G matrix. A moments fit retains the unconstrained covariance component

estimates. With one block, the component contains the selected matrix directly. Trait selection retains every coefficient of each selected trait, in coefficient order within the trait. The grouped result omits blocks without a selected coefficient. A single selected block requires at least one matching coefficient.

residual.scale The square root of the residual variance multiplier when there is one univariate, one-dimensional residual covariance block. Otherwise NA_real_. With precision weights or a residual basis, the multiplier need not equal every observation's residual standard deviation; R gives the observation-level residual covariance.

ngroups Named counts of the levels represented in each random covariance block, including extra levels supplied through a known covariance matrix. The unit attribute distinguishes group levels from response cells in direct covariance-kernel blocks.

joint The stored joint fixed/random prediction-error covariance matrix, or a zero-by-zero matrix when it was not stored. Explicitly requesting name="joint" reconstructs this matrix at the fitted covariance parameters when necessary for ML and REML fits. Reconstruction holds the fitted parameters fixed and leaves object unchanged. In a moments fit this component is a zero-by-zero matrix.

joint.order A data frame declaring every position in joint: position, a unique label, kind (fixed or random), term, level, coefficient, fixed.column, block and group.index. Inapplicable entries are NA. The fixed positions are independent columns in the engine's pivot order. In a moments fit this data frame has zero rows.

estimable A list with logical fixed flags for individual coefficients in the original parameterization, the independent fixed.columns, the fixed.map relating original columns to the independent design, and logical fitted flags for response-cell means. A coefficient can have a finite representative estimate but fail to be individually estimable in an aliased parameterization. Fitted flags are NA for rows omitted because model predictors are unavailable. The scope entry explicitly separates these estimability flags from fit convergence and covariance-parameter identification.

metadata The method, formulas, response traits, observed response-cell indices, omission mask, offsets, weights, fixed rank and aliased column names, convergence state and engine diagnostics, available covariance-parameter information rank and status, joint-covariance storage status and interpretation, matrix ordering, and fitted-object version. The availability entry describes the status of random predictions, joint covariance, its ordering and G. Unavailable covariance-parameter information is reported as such. Information rank deficiency is distinct from fixed-column aliasing and from convergence failure. It can also reflect a boundary parameterization, so covariance.parameters$confounded is NA: these extractors do not separately diagnose covariance-parameter confounding.

Additional named components are available:

series The second-stage inputs and target declaration for a series fit. The data data frame and rcov matrix are the retained second-stage inputs. Their row names and any covariance-axis labels are preserved. The matrix is the inherited or supplied target covariance on the response scale. The cells data frame has the complete target labels, trial, genotype, trait and data-row mapping before second-stage row selection. The reference entry contains the declared grids and weights, or NULL for automatic targets. The genotype entry identifies the resolved genotype column. The covariance.provenance entry records an explicitly supplied joint target covariance and is NULL under the default independence assumption. The independence entry records the earlier fits, observation counts and disjointness check. A fit without a second-stage series declaration gives an error.

`variance` A matrix with one row per covariance block and one column per trait, including when a single block or trait is selected. The entries are diagonal elements of the fitted coefficient covariance matrices. A trait with no coefficient in a block has an `NA` entry. A selected trait with several coefficients in a block requires the `covariance` component to obtain their separate variances. For a `homdiag` block with a common coefficient variance, the matrix gives that shared variance.

`correlation` The correlation matrices of the fitted coefficient covariance blocks, with the same structure and selection rules as `covariance`. A coefficient with a nonpositive or nonfinite variance has `NA` correlations. Positive variances are standardized using the fitted covariances, including unconstrained moments estimates.

`observations` Counts of retained data rows and observed response cells, overall and by trait, the fixed-design rank and residual DF, and the row and cell indices. Moments fits also provide the residual error stratum DF by trait.

`tests` A table of the stored fixed, random and joint tests in a common format, including their methods, statistics, DF and p values. The joint rows of `anova(object, effects = "all")` include the MANOVA statistics and their F reference calculations.

`sscp` A list of sums-of-squares-and-products results for method-of-moments fits. `H` is the list of MANOVA hypothesis matrices named by model term, and `E` is the error matrix for the original fit. `strata` and `pooled` contain the original and pooled orthogonal source matrices, respectively, named by source. Matrices have traits on both axes, including one-by-one matrices for a single trait. An unavailable matrix has zero rows and columns, and an unavailable list has zero elements.

`sscp.H` The stored hypothesis sums-of-squares-and-products matrix for `term`, with the trait names on both axes. A term without a stored hypothesis matrix gives an error.

`sscp.E` The error sums-of-squares-and-products matrix for the original fit, with the trait names on both axes. A univariate error matrix is one by one. A fit without an error matrix gives a zero-by-zero matrix.

`predictions` The stored hybrid prediction data frame with two parental identifier columns named for the fitted parental pools and an integer `source.row` column. For `prediction = "NAonly"`, `source.row` identifies the row in the retained performance data. For `prediction = "Full"`, the parental identifiers specify the cross and `source.row` is `NA`. The table has a numeric `prediction` column for one trait, or a `prediction.<trait>` column for each fitted trait. For `"NAonly"`, an observed trait cell has no cached prediction and its entry is `NA`. Extraction reuses the saved predictions. A non-hybrid fit returns a zero-row data frame with factor columns `parent1` and `parent2`, integer `source.row` and numeric `prediction`.

The following dense matrices are returned only when explicitly named:

`X` The complete fixed-effects design, including aliased columns.

`Z` The combined random-coefficient design.

`G` The full covariance of the random coefficients. For an ordinary block with coefficient covariance S and known level covariance A, its block is $S \otimes A$; independent levels use the identity matrix for A. A direct kernel block uses an identity design in `Z` and its fitted kernel covariance in `G`. Moments fits with random terms return a zero-by-zero matrix for `G`. Their component estimates are available in `covariance`.

`R` The residual covariance between response cells, including all residual blocks, their bases, any level covariances, and precision weights.

These matrices can be large. The standard component list does not construct `X`, `Z`, `G` or `R`; explicitly requesting an uncached `joint` matrix also requires dense storage for that matrix.

Ordering, units and uncertainty: The rows of `X`, `Z` and `R` follow the retained model's response-cell order, with all rows of the first trait before all rows of the next trait for a multivariate response. The `observed` indices in `metadata` identify the response cells used in fitting. Matrix rows can therefore include cells without an observed response. Offsets are reported separately in `metadata$offset`; they are not columns of `X`.

Within a random covariance block, coefficient positions vary first over all group levels of the first coefficient, then all levels of the next coefficient. Thus they match the column-major vectorization of the corresponding matrix in `random`. Blocks remain separate even if they refer to the same grouping variable. `Z`, `G` and the random positions of `joint` share this order. Matrix names and `joint.order` declare it explicitly.

With covariance parameters held at their fitted values, `joint` estimates

$$\operatorname{Cov}\left[\begin{pmatrix}\widehat{\beta}-\beta\\ \widehat{u}-u\end{pmatrix}\right].$$

The random block is prediction-error covariance with fixed effects estimated. It includes the contribution of fixed-effect estimation and the fixed/random cross covariance. It is not the conditional covariance calculated with fixed coefficients treated as known, nor does it include covariance-parameter uncertainty or residual variation of a future observation. Only independent fixed columns occur in `joint`; `estimable` supports interpretation of an aliased fixed model. The returned matrix carries `uncertainty.kind` and `includes.covariance.parameter.uncertainty=FALSE` attributes.

Numerical rank and principal components: For each random-coefficient covariance matrix S_k, the principal component calculation returns its eigenvalues in descending order and an orthonormal eigenvector matrix. Both covariance and eigenvalues use the original coefficient units. This calculation concerns S_k, not the rank of a supplied level covariance matrix A_k. A singular supplied relationship matrix alone does not make `st.lmm.singular` return TRUE.

The quantity $\bar{r}$ is the mean diagonal of R over observed response cells, and $\lambda_{k,1}$ is the largest eigenvalue of S_k. The reported reference variance is $s_k = \max(\bar{r}, \lambda_{k,1})$. The numerical rank is the number of eigenvalues satisfying $\sqrt{\lambda_{k,j}/s_k} > \mathrm{tol}$. If the reference variance is zero, every relative standard deviation is defined to be zero. Negative eigenvalues within numerical roundoff are truncated to zero; a materially indefinite covariance matrix is an error.

This definition is unchanged by a common rescaling of the responses and is specified in the original random-coefficient basis. Changing predictor units can change a relative-rank assessment. The definition differs from a boundary check on optimizer or Cholesky coordinates. The eigenvalues, reference variances, tolerances and ranks are returned so the criterion can be inspected directly.

An eigenvector's sign is arbitrary. Repeated eigenvalues identify a principal subspace rather than unique eigenvectors. Comparisons should use eigenvalues and subspaces instead of requiring identical eigenvector signs or bases. Fixed-column aliasing, covariance-parameter information deficiency and lack of convergence are reported separately and do not themselves set the singularity flag. The convergence attribute should be considered when interpreting a diagnostic from an unfinished fit.

Value

`st.lmm.components` returns the value of a single named component. Multiple names or a NULL request return a named list of components. Matrices retain their dimensions, including one-row

and one-column cases. Extracted values retain the package's result-display class in addition to their ordinary R structure.

`st.lmm.pca` returns a named list with one entry per random covariance block. Each entry contains `sdev`, `rotation`, `eigenvalues`, `relative.sdev`, `rank`, `dimension`, `reference.variance`, `tol` and `singular`. The list has `tol`, `rank.definition`, `mean.residual.variance` and `converged` attributes. A model without random blocks returns an empty list with these attributes.

`st.lmm.singular` returns one logical value: TRUE if any random coefficient covariance block has numerical rank smaller than its dimension. Named attributes `terms`, `rank`, `dimension`, `eigenvalues`, `relative.sdev` and `reference.variance` give the block-level results. It also carries `tol`, `rank.definition`, `mean.residual.variance` and `converged`. A model without random blocks returns FALSE.

See Also

`st.lmm`, `st.predict`, `st.lsm`

`st.lmm.diagnostics` *Leverage and Residual Diagnostics for Gaussian Mixed Models*

Description

Calculates marginal or conditional residuals, residual variances, standardized residuals and leverage at the fitted covariance parameters. The diagnostic table retains explicit response-cell, row and trait keys.

Usage

```
st.lmm.diagnostics(object, type = "conditional")
## S3 method for class 'st.lmm'
hatvalues(model, type = "conditional", ...)
## S3 method for class 'st.lmm'
rstandard(model, type = "conditional", ...)
```

Arguments

`object`, `model`	An `st.lmm` object fitted by ML or REML. Univariate and multivariate responses are supported. The `hatvalues` and `rstandard` methods also support univariate method-of-moments models containing only fixed effects.
`type`	`"conditional"` includes fitted random effects in the mean. `"marginal"` uses the fixed mean, including fitted offsets.
`...`	Additional arguments to the generic; currently unused.

Details

All quantities condition on the fitted covariance parameters. They do not add covariance-parameter estimation uncertainty or apply Kenward–Roger adjustments. The result records whether the original covariance fit converged; evaluating diagnostics does not make an unconverged fit valid.

The covariance of the observed response cells is $V = G + R$, where G is the contribution of all random effects and R is the residual covariance. The matrix X contains an independent basis for the fixed mean, with $C = (X'V^{-1}X)^{-1}$ and $P = V^{-1} - V^{-1}XCX'V^{-1}$.

Marginal leverage is the diagonal of $H_m = XCX'V^{-1}$. Conditional leverage is the diagonal of $H_c = I - RP$. With correlated residuals these matrices need not be symmetric; leverage values are not truncated to the interval from zero to one.

The conditional residual is the observed response minus its fitted fixed and random mean, with variance given by the diagonal of RPR. The marginal residual subtracts only the fixed mean and offset, with variance given by the diagonal of $V - XCX'$. `rstandard` divides the selected residual by the square root of its variance. A zero variance gives `NA` and an explicit status rather than a zero standardized residual.

For a univariate fixed-only method-of-moments model, `hatvalues` uses the weighted fixed-design leverage and `rstandard` uses the ordinary regression residual variance with DF equal to the number of observations minus the fixed-design rank. Positive precision weights multiply the response residual by their square root before standardization. Marginal and conditional results agree in this case. The `st.lmm.diagnostics` table remains available only for ML and REML.

`residuals(object)` uses the conditional response residual for a likelihood mixed model and the ANOVA projection residual for a mixed moments fit. The corresponding explicit types are `"conditional"` and `"projection"`. For a model without random effects, the default is the residual from the fixed mean. `residuals(object, type="marginal")` subtracts the fixed mean. For ML and REML, `type="pearson"` divides the conditional response residual by the square root of the diagonal of R. Pearson residuals and `rstandard` generally use different denominators.

`plot(object, which=1:4)` displays residuals versus fitted values, a normal Q–Q plot of standardized residuals, a scale-location plot, and standardized residuals versus leverage. Character selectors are `"residuals"`, `"qq"`, `"scale-location"` and `"leverage"`. A model with subset-selection results retains its subset inclusion plot by default; `which` requests a diagnostic panel. Plots are descriptive, since residuals can be correlated. They report unavailable cells, annotate at most `id.n` observations, and invisibly return their exact numeric plot data. Plotting does not change the graphics layout; the usual graphics controls determine a multi-panel page.

Value

`st.lmm.diagnostics` returns a data frame with columns `cell`, `row`, `original.row`, `row.name`, `trait`, `observed`, `estimable`, `response`, `fitted`, `residual`, `variance`, `standardized`, `leverage` and `status`. It describes retained response cells, including missing responses. The original-row key refers to the row of the data supplied to the fitted model; it can repeat when the fitted subset contains repeated rows. Attributes state the diagnostic type, covariance assumption, leverage and variance definitions, and convergence status.

`hatvalues` and `rstandard` return matrices with one row per observation and one column per trait. They preserve row and trait labels, and restore omitted selected rows with `NA` when the model uses `na.exclude`. Undefined or nonestimable quantities remain `NA`. The diagnostic table itself uses retained-row keys and is not expanded by `na.exclude`.

See Also

st.lmm, st.lmm.components, influence.st.lmm, rstudent.st.lmm

---

st.lmm.model *Model Information and Updates for Gaussian Mixed Models*

Description

Extracts retained model information or constructs a new fit from an st.lmm object. The original fitted object is not modified.

Usage

```
## S3 method for class 'st.lmm'
model.frame(formula, ...)
## S3 method for class 'st.lmm'
terms(x, ...)
## S3 method for class 'st.lmm'
weights(object, ...)
## S3 method for class 'st.lmm'
update(object, formula., ..., evaluate = TRUE)
```

Arguments

formula, x, object	An st.lmm fitted object.
formula.	An optional formula update, such as . ~ . + x, processed by update.formula. The named argument fixed in ... can instead replace the entire fixed formula.
evaluate	A single nonmissing logical value. TRUE fits the updated model; FALSE returns the call without evaluating additional arguments.
...	For update, named arguments to st.lmm. For the extraction methods, reserved for compatibility with the generics.

Details

The retained model frame contains the response, evaluated fixed-formula variables, raw variables used by random and residual formulas, and supplied weights and argument offsets. Fitted factor levels and contrast matrices are retained. The terms object records the fitted prediction recipes for transformations such as poly and scale. model.response, model.offset and model.weights can extract their respective components from the returned frame. Matrix responses remain matrices.

Under the default missing-response policy, a row with a missing response can remain a prediction row. Its presence in model.frame does not imply that it contributed to the likelihood. The fitted object's model$input.rows and model$row.names retain row identities; nobs reports the observed response count used by the fit. Under explicit na.omit or na.exclude, the frame contains only retained rows; the latter pads public response-level extraction without padding the frame or engine matrices. The frame's na.action attribute identifies removed positions in the selected row order.

The model$input record keeps the original data snapshot, selected.rows (original row numbers, with repetitions allowed), selected.row.names, and retained.positions within that selected layout. model$input.rows gives the retained original row numbers. na.removed, weight.removed and common.complete.removed record distinct removal stages. prediction.omitted identifies retained rows whose predictors were unusable under the default policy. These maps refer to data rows; a multivariate likelihood may use several response cells per retained row.

An update starts from the original input snapshot and fitted settings, then applies its subset and missing-value policy again. It does not subset an already selected frame. A changed formula can therefore recover rows whose previously missing variables are no longer needed. Changing or removing the caller's original data binding therefore does not change a subsequent update. Supplying a new data argument explicitly uses the new data. Data-derived subset, weight and offset expressions are reevaluated there. An external training vector for subset, weights or offsets requires an explicit replacement when changing data; it is not silently recycled or matched only by length. Formula environments continue to resolve functions and external objects referenced by the formulas; callers should keep such objects available or include the required variables in the data frame.

Existing fitted factor coding is retained unless replaced through contrasts. If an updated subset removes factor levels, the saved contrast rows are matched to the remaining levels while retaining the fitted contrast columns; resulting aliases are reported. Additional arguments are kept as expressions until the returned call is evaluated, so data-dependent arguments such as weights = precision are evaluated by st.lmm in its usual data environment. The returned call uses the package-qualified fitting function and can be evaluated after saving and reloading the model.

When the fitted object contains a subset search, an update changes the specified full model. The search is repeated only when selection is supplied explicitly; a warning explains this behavior. No selected candidate automatically replaces the fitted model.

Value

model.frame returns a data frame with a terms attribute; terms returns a terms object; weights returns the retained numeric precision vector, with zero elements for an unweighted fit. update returns a new st.lmm object, or an unevaluated call when evaluate = FALSE.

See Also

st.lmm, model.frame, update

---

st.lmm.profile *ML Likelihood Profiles for Gaussian Mixed Models*

Description

Constructs likelihood profiles for fixed coefficients and supported random and residual standard deviations and correlations. All other estimable parameters are re-estimated at each specified target. Confidence limits and signed-root likelihood-ratio plots use the resulting ML profile, including an ML refit when the original model used REML.

Usage

```
st.lmm.profile(object, parm = NULL, level = .95, control = list())

## S3 method for class 'st.lmm'
profile(fitted, ...)

## S3 method for class 'st.lmm.profile'
confint(object, parm,
    level = object$level, ...)

## S3 method for class 'st.lmm.profile'
print(x, ...)

## S3 method for class 'st.lmm.profile'
plot(x, which = NULL, ...)
```

Arguments

object, fitted	An st.lmm model fitted by ML or REML for st.lmm.profile and profile; an st.lmm.profile result for its confidence-interval method. Method-of-moments fits have no likelihood profile and are rejected.
parm	Parameter names or integer indices. For st.lmm.profile, NULL selects all supported parameters. parm = character() is available to obtain the parameter inventory without tracing any curves; its parameters component gives names, estimates, physical bounds and identifiability. For confint applied to a profile result, a missing parm selects all previously profiled parameters, and indices refer to that selected set. Duplicate and unknown parameters are errors.
level	A confidence level strictly between zero and one. Requesting a different level from confint on a saved profile retraces the requested parameters at that level.
control	Named list of profile controls. The defaults are maxit = 1000, maxsteps = 24, maxroot = 60, maxrestarts = 8, score.tol = 1e-7, cone.tol = 1e-7, lr.tol = 1e-5, constraint.tol = 1e-10, and extent = 1e6. maxit limits the initial nuisance-optimization iterations per start and restart; a subsequent Newton refinement can take up to 35 steps. maxit = 0 evaluates the supplied starts without optimization. maxsteps limits outward bracketing steps per side; maxroot limits its endpoint refinement; maxrestarts limits restarts from feasible covariance-boundary directions. score.tol and cone.tol bound the nuisance projected score and covariance-cone violation. lr.tol bounds the error in the endpoint likelihood-ratio statistic. constraint.tol bounds the relative error in the fixed physical target. These four tolerances may be tightened, but not relaxed beyond their defaults. extent limits the absolute search distance to its value times max(1, abs(estimate)). Unknown controls are errors.
x	An st.lmm.profile result.
which	Names or indices of profile curves to plot. NULL plots all selected curves in their stored order.
...	For printing, display controls whose omitted values use st.options. Arguments passed from profile to st.lmm.profile, or named graphical arguments

for `plot`. The profile `print` and `confint` methods accept the generic's remaining arguments but do not use them.

Details

Likelihood profiles for separable covariance products, including the established AR1 by AR1 family, and US trait-by-occasion products are not implemented. Such requests raise an explicit error; no unconstrained covariance model is substituted.

Repeated fits run in the current R process. Numerical kernels use the existing OpenMP and BLAS thread controls; `st.set.num.threads` describes these controls. R callbacks execute on the main R thread. Intermediate state stays in memory.

Parameter names begin with `beta:`, `sd:`, or `cor:`. The remaining labels identify the covariance block and its coefficient or coefficient pair. Standard deviations are marginal standard deviations in the original coefficient units. They are not Cholesky coordinates or standard deviations divided by the residual scale.

The inventory includes estimable fixed coefficients, residual standard deviations, independent or unstructured random-effect standard deviations, and every pairwise correlation of an unstructured random block. Homogeneous random or residual covariance structures contribute one common standard deviation; heterogeneous structures contribute one per coefficient or occasion. Compound-symmetric and AR1 random or residual covariance structures contribute their common correlation parameter, subject to the corresponding dimension-dependent or unit bounds. Fixed covariance blocks do not contribute estimated parameters. Pairwise correlations of an unstructured residual block are optimized as nuisance parameters but are not available as individual profile targets.

The calculation first obtains a converged ML baseline with the retained response, observation mask, fixed and random designs, precision weights, offsets, and covariance specification. This uses the retained resolved model also for SelectionTools marker and hybrid data sets. The original fitted object is unchanged. The result records both `method = "ML"` and `original.method`; REML-based small-sample degrees of freedom are not used to calibrate the profile. `confint(fit, method = "profile", parm = ...)` provides a direct interval interface. The existing `confint(fit)` default remains the Wald interval for fixed coefficients.

Every curve uses the same ML baseline. Numerical records retain warnings and failed nuisance-optimization starts. A materially better ML optimum invalidates the requested intervals; numerical details remain available for diagnosis.

For a parameter value t, the likelihood-ratio statistic is twice the ML log-likelihood difference between the baseline and the constrained fit. The confidence limit solves the equation equating this statistic to the `level` quantile of a chi-squared distribution with one degree of freedom. Its signed square root is negative below the estimate and positive above it. This likelihood-ratio calibration is approximate in small samples and at variance or correlation boundaries.

Each constrained fit compares the preceding successful point, the fitted baseline, and a cold start. Positive-semidefinite covariance constraints and the selected target are preserved throughout nuisance optimization. Convergence requires the nuisance score and feasible covariance-direction checks, as well as the target constraint. Tiny free structured standard deviations may be projected to zero as a numerical restart only when the objective change is within a recorded floating-point roundoff bound and the target is preserved. All convergence checks are then repeated. The projections and every start are retained in the result. A critical-cone calculation involving more than 200 nuisance covariance coordinates, or an unresolved degenerate cone within its work limit, yields an

incomplete point rather than an accepted interval. The same conservative result applies to a partial-zero heterogeneous AR1 residual block when its correlation is unidentified and boundary curvature over the equivalent correlations remains uncertified.

A pairwise correlation of an unstructured random block is unidentified when either associated standard deviation is zero. The shared correlation of a compound-symmetric or AR1 block requires at least one informative pair with positive standard deviations. For residual blocks, those positions must also be observed together within a group. If informative AR1 pairs have only even lags, the sign of the correlation is unidentified. Aliased fixed coefficients are also marked unidentified. These requests remain visible in the output and do not yield ordinary confidence limits.

The endpoint statuses distinguish `complete` (the likelihood-ratio threshold was reached), `boundary` (the physical bound was reached within the accepted set), `flat` (approximately flat over the examined range), `open` (the search limit was reached without a bracket), `unidentified`, and `failed`. An open search does not by itself establish an infinite confidence limit. `confint` returns `NA` for an endpoint that is not `complete` or `boundary`, and attaches a matrix of endpoint statuses. A constrained start that achieves a materially better ML likelihood than the baseline triggers a warning and invalidates all requested intervals; all start records remain available for diagnosis. A lower objective from an unconverged start cannot be silently discarded in favour of a worse converged start.

Profile plotting uses the ordinary graphics layout and does not change `par(mfrow)` or force a multi-panel arrangement. Each requested curve is plotted with horizontal reference lines at the confidence threshold.

Value

`st.lmm.profile` and `profile` return a serializable object of class `st.lmm.profile`. Its principal components are:

- `parameters`: the complete physical parameter inventory.
- `selected`: names of the requested parameters.
- `intervals`: estimates, endpoints, endpoint statuses, likelihood-ratio errors, searched values and point identifiers.
- `profiles`: a concise point table and detailed start records for each selected parameter. The detailed records contain physical covariance matrices, fixed coefficients, nuisance scores, covariance-cone diagnostics, constraint errors, numerical coordinates and their exact constrained map, and any accepted boundary projections.
- `baseline`, `baseline.deviance`, `method`, and `original.method`: the ML reference fit and its provenance.
- `control`, `level`, `cutoff`, `complete`, and `diagnostic`: the effective settings and overall completion status.
- `object`: the original fitted model, retained for retracing.

The confidence-interval method returns a two-column numeric matrix with `status`, `method` and `level` attributes. The print and plot methods return their profile object invisibly.

See Also

`st.lmm`, `st.lmm.refit`, `st.lmm.objective`, `confint`

st.lmm.refit *Refits a Retained Gaussian Model and Evaluates Its Likelihood*

Description

An ML or REML model is refitted with replacement responses, observation precision weights, or an estimation method. The fixed-effect design, random-effect bases, grouping indices, known covariance matrices, factor contrasts and offset values are retained. A separate interface evaluates the Gaussian likelihood and its covariance-parameter gradient without fitting a model.

Usage

```
st.lmm.refit(object, response, weights, method = NULL,
             control = list(), warm = TRUE)

st.lmm.objective(object, parameters = NULL, method = NULL,
                 gradient = TRUE)
```

Arguments

object	An st.lmm object fitted by ML or REML.
response	Optional replacement response on the original response scale, before subtraction of offsets. A numeric matrix has one row per retained observation and one column per trait. A univariate model also accepts a numeric vector. Values are matched by position and the missing-value pattern follows model.response(model.frame(object)). Omitting this argument retains the response; explicit NULL is an error.
weights	Optional finite, strictly positive numeric precision weights, one per retained data row. This argument is supported for univariate models. Omitting it retains the fitted weights; explicit NULL removes them on the retained rows.
method	NULL to retain the fitted method, or "ML" or "REML".
control	A named list overriding the retained fitting controls. Further details appear in st.lmm.
warm	Logical. TRUE uses the fitted covariance matrices, in their original units, as starting values. FALSE uses the usual default starts for estimated covariance blocks. Fixed covariance blocks are retained.
parameters	A numeric vector in the coordinate system reported by st.lmm.objective(object). NULL uses the fitted parameters. If names are supplied, they must match the reported names and order exactly.
gradient	Logical. A value of TRUE includes the gradient of minus twice the log likelihood with respect to the reported parameters.

Details

Refitting reuses the saved model representation and does not reevaluate the original data binding, predictor expressions, subset, or missing-value action. The parent fit and supplied response and weight vectors remain unchanged. Rows excluded when the parent model was fitted cannot be restored by refitting. This also applies to rows excluded by zero ordinary regression weights. Missing-response rows retained for prediction remain in the refitted model; replacement responses must preserve their missing values. Changes to the observation layout or model specification require `st.lmm` or `update`.

Precision weights modify only the residual covariance. Its row and column i are scaled by $1/\sqrt{w_i}$; random-effect covariances retain their original meaning. A response replacement is the evaluated response: a formula such as `log(y) ~ x` requires new log responses. Offsets are subtracted internally. The evaluated replacement response is retained through serialization and through updates using the same response expression and data snapshot. An explicit new data argument or a changed response expression starts a new response evaluation.

Replacement weights, including explicit removal by `weights = NULL`, are retained through serialization and updates using the same data snapshot. They are matched to the retained row layout, including repeated rows created by a subset. An explicit `weights` argument to `update` overrides these saved weights. Replacement data after a numeric weight refit require explicit weights for the new data. Previously removed weights remain removed when replacing data unless new weights are supplied.

Changing a Kenward–Roger fit to ML produces a warning and uses asymptotic inference, because Kenward–Roger inference requires REML. Other inference settings are retained. A model fitted with subset selection is refitted as its full fitted model, with a warning; the subset search is not repeated.

Warm starts transfer covariance matrices in physical units. Optimizer coordinates are scaled using the current response and residual bases and can therefore change after replacing responses or weights. Previously stored `control$parameters` are discarded. Explicit parameters in the new `control` list override the warm start and must use the new fit's coordinate system. For structured random-effect covariances, the direct correlation coordinate is retained as well. This preserves its sign even when a zero variance makes the fitted covariance matrix insufficient to identify that sign.

The objective is minus twice the full Gaussian ML or REML log likelihood, including its normalization constant. Fixed effects are profiled out; covariance scale remains an explicit parameter. The function does not optimize parameters, change the model, or return an external pointer or a stateful function. Its numeric record can be serialized or used by an external optimizer. Parameters outside the stated bounds are rejected. A point satisfying the parameter bounds but failing to define a valid marginal covariance returns `value = Inf`, `valid = FALSE` and an `NA` gradient.

The coordinate table describes each variance, common variance, standard deviation, Cholesky entry, log variance, log standard deviation, direct correlation, or partial-correlation coordinate. Its physical value is the reported `scale` times the function in `inverse` applied to the parameter: `identity`, `exp`, or `tanh`. The `covariance` component gives the resulting covariance matrices in the original coefficient units. Coordinates are specific to the retained model and must be obtained again after a response or weight refit. Row and column zero denote a block-wide common variance or correlation. Compound-symmetry correlation bounds depend on the number of coefficients; AR1 correlations have bounds minus one and one. These direct correlations use `identity`, rather than a Fisher transformation.

Fixed/shared physical covariance constraints are retained by refitting and recompiled for the response and weight scaling of the new fit. Warm and default starts are reconciled with the retained declarations. The objective accepts the complete raw coordinate vector as before, but a supplied vector must satisfy every retained fixed/shared constraint. Violations are explicit errors; arbitrary evaluation vectors are not silently projected.

The additional `constraints` record reports canonical physical SD and correlation selectors, variance aliases, physical values, fixed/free/shared status, complete and independent parameter counts, and each raw coordinate's independent index. `free.parameters`, `free.coordinate` and `free.gradient` describe the reduced optimizer coordinates. In an objective result, both reported gradients differentiate the same minus-twice- log-likelihood value. A shared coordinate's reduced derivative is the chain- rule sum over its members; fixed coordinates do not contribute a free direction. The original-scale constraints are distinct from the complete normalized optimizer coordinates. The `st.lmm` help page describes the structures and operations.

Value

`st.lmm.refit` returns a complete `st.lmm` object with recomputed estimates, covariance matrices, tests and predictions, together with a `refit` record describing the replacement arguments and initialization.

`st.lmm.objective` returns a list containing `value`, `gradient` (or `NULL`), named `parameters`, named `lower` and `upper` bounds, the `coordinates` data frame, physical `covariance` matrices, `method`, `nobs`, fixed-effect `rank`, a `valid` flag, a description of the likelihood scale, and the physical `constraints` inventory and reduced-coordinate metadata. The coordinate table identifies the covariance block, its type and structure, coefficient row and column, transformation and physical scale. For separable products there is one common variance coordinate and one `rho.axis` coordinate for each correlated axis. Axis-specific CS bounds are retained. Independent axes add no correlation coordinate. Refitting retains the product layout and direct-sum strata. For US trait-by-occasion products, the trait coordinates are `trait.log.sd` and `trait.partial.correlation.z`, followed by `rho.occasion` for AR1 or CS. Trait-coordinate row and column indices refer to traits, not expanded trait-by-occasion coefficients. Warm starts transfer the complete physical product matrix; cold starts preserve the family and declared occasion grid. Missing trait cells and missing occasions are retained under the fitted missing-data rules. Explicit covariance constraints are not available for this US family.

See Also

`st.lmm`, `st.lmm.components`, `update`, `optim`

---

st.lmm.simulate *Simulates Gaussian linear mixed-model responses*

Description

Generates responses from a fitted likelihood model or from a fully specified Gaussian model without fitting it. Selected random coefficients can be held at their fitted values while the other coefficients and residual errors are regenerated.

Usage

```
## S3 method for class 'st.lmm'
simulate(object, nsim = 1, seed = NULL, re.form = NA,
    conditional = NULL,
    newdata = NULL, newparams = NULL, allow.new.levels = FALSE,
    offset = NULL, weights = NULL, ...)

st.lmm.simulate(fixed, random = NULL, cov = ~units, data, parameters,
    nsim = 1, seed = NULL, weights = NULL, offset = NULL, ucov = NULL,
    contrasts = NULL, traits = NULL, ...)
```

Arguments

`object`	An `st.lmm` object fitted by ML or REML. Simulation from the moments method is not supported.
`nsim`	A positive integer giving the number of response replicates.
`seed`	`NULL`, or a finite integer passed to `set.seed`. An explicit seed is used within the call and the caller's previous random number state, including its absence, is restored. Seeded calls reject Box–Muller and user-supplied generators because their complete state is not stored in `.Random.seed`. With `NULL`, the call advances the caller's current stream without restoring it; these generators are then allowed, but replay from the returned seed alone is not guaranteed.
`re.form`	Random coefficients to hold at their fitted values. The default `NA`, `~0`, `~-1`, or `character(0)` regenerates all random coefficients. `NULL` holds all random coefficients. A character vector selects exact complete covariance-block names. A one-sided random formula can select fitted coefficients within a block: for example, `~1|group` holds only the random intercept from a fitted `~1+x|group` term. Formulas combining several selected terms are supported. Names and coefficient definitions must match the fitted model.
`conditional`	Optional string convenience policy. `"none"` is exactly equivalent to `re.form=NA` and regenerates all random coefficients; `"all"` is exactly equivalent to `re.form=NULL` and holds all fitted random coefficients. `NULL` preserves the established `re.form` argument and default.
`newdata`	A data frame or named list containing target model variables, or `NULL` for the retained training rows. Its response column is not used. Fixed predictors, needed random predictors, residual strata and any residual occasion and grouping variables must be available and nonmissing. Numeric model variables must be finite. Learned transformation and contrast definitions are retained from the fit.
`newparams`	Optional physical parameters, in the layout described below. Omitted top-level entries use fitted values. No covariance optimization is performed.
`allow.new.levels`	Whether new levels of an independent random grouping factor are allowed. They have zero fitted random means and their specified random covariance when regenerated. A named known covariance matrix must already contain the target levels, or the fitted hybrid covariance must define their parental relationships.

`fixed` For `st.lmm.simulate`, a one-sided fixed-effects formula. There is no observed response and no response expression is evaluated. `~0` specifies a model with no fixed coefficients.

`random, cov` Random-effect and residual covariance formulas, using the same syntax and covariance families as `st.lmm`. Homogeneous factor products and `dsum(~covariance | stratum, group=Subject)` use the fitted axis order and independent-block structure. Supplied covariance matrices must obey the product shape; arbitrary positive-semidefinite matrices are not interchangeable with that family. Univariate parental overlays support ordinary grouping interactions and selected `at()` strata, retaining shared parent identities, self-cross loading two, and a common parental relationship matrix. Independent grouping levels use separate parental vectors; an `at()` block retains its own variance. Overlay coefficient formulas, correlated coefficient products and multivariate overlays remain unsupported.

`data` A named data frame or list of model variables. The same bitpacked SelectionTools marker and hybrid data-set inputs accepted by `st.lmm` can also supply these variables and known covariances. supported names must be quoted. Supported specifications include `ucov=list(gen="A")` and the hybrid product `"kro(a,a)"` for symbolic covariances; bare matrix names evaluate R objects. The `random` interface also accepts `||`, equivalent to a diagonal coefficient covariance with `|`. No observed response is required.

`parameters` Complete physical parameters for `st.lmm.simulate`. Both the named fixed-coefficient vector and all random and residual covariance blocks must be supplied. Further details appear in Details.

`weights` For `st.lmm.simulate`, one finite, strictly positive residual precision weight per row, evaluated in `data` and then the formula environment. For fitted simulation, an explicit override of future residual precision requires `newdata`; `weights=NULL` removes target weighting. Training-row simulation retains the fitted weights. A data-derived fitted weight expression can be reused for new data; an external vector or a retained-weight refit requires explicit target weights. Weights currently require a univariate response.

`offset` For `st.lmm.simulate`, a numeric scalar or one value per row, evaluated in `data` and then the formula environment. For fitted simulation, an explicitly supplied value replaces only the argument-offset part; `NULL` removes that part. Formula offsets are added once. An omitted value reuses the fitted offset or its stored new-data recipe. An external training offset vector requires an explicit override for new data. Offsets currently require a univariate response.

`ucov` Named known covariance matrices or supported SelectionTools covariance requests, as for `st.lmm`.

`contrasts` Optional factor contrasts, as for `st.lmm`.

`traits` `NULL` for a univariate response, or unique, nonmissing, nonempty trait names. Multiple names define a multivariate response without requiring an observed response matrix.

`...` Additional arguments are rejected. In particular, `st.lmm.simulate` has no fitted random coefficients to hold and does not accept `re.form`.

Details

Multivariate residual products such as `cov=~us(trait):ar1(Visit) | Subject` use one unstructured trait covariance and one unit-diagonal occasion correlation matrix. Homogeneous CS and homdiag occasions are also supported. The required input is the full physical trait-by-occasion covariance matrix, with occasion varying fastest inside trait and matching coefficient names. The matrix must obey the requested product form; a general positive-semidefinite matrix is insufficient. Declared occasion levels, including absent levels, retain their lag spacing. Singular positive-semidefinite products are supported for specified simulation. Exact linear relations in such draws can differ by ordinary floating-point eigendecomposition roundoff. Further details appear in `st.lmm` for the residual-only syntax and current composition restrictions.

The simulation model is $y = X\beta + o + \sum_k Z_k u_k + e$. Fixed coefficients and covariance parameters are treated as specified values. No uncertainty in their estimation is drawn. Unless selected by `re.form`, random coefficients have zero mean and their specified block covariance; residual errors have zero mean and the specified residual covariance, including precision weights, residual heterogeneity and the structured residual families described in `st.lmm`. With repeated-observation residuals, occasion levels and their order are retained from the fitted model, even when `newdata` contains only a subset of them. New levels of the residual grouping factor are allowed; new occasion levels and residual strata without fitted covariance parameters are errors. AR(1) gaps retain their fitted factor-level distances.

The meaning of `re.form` differs from prediction: it selects random coefficients to *hold*, rather than components to include in a predicted mean. Held coefficients are plug-in constants equal to their fitted values. The remaining coefficients are regenerated from their *marginal* covariance. If an intercept is held while a correlated slope is regenerated, the slope retains its marginal variance, with no intercept–slope covariance in the simulated response. This is not a conditional Gaussian draw given the held coefficients and is not a draw from their posterior distribution.

Physical parameters have the following layout:

```
list(fixed = c("(Intercept)" = 2, x = 0.5),
     covariance = list(random = list(group = S),
                       residual = list(units = E)))
```

The names of `fixed` identify every fixed model-matrix column exactly once. In multivariate models these names have the form `trait:coefficient`, in trait-major order. Supplied values must be finite, including positions aliased in a fitted model; `numeric(0)` is used when the model has no fixed coefficients. Every supplied covariance section names all blocks in that section. Block names match the fitted model, and each block is a matrix whose row and column names match its coefficient axes. A scalar variance is therefore a named one-by-one matrix, not a standard deviation. Names can be reordered and are aligned explicitly. The `fixed` and `covariance` entries of `st.lmm.components(object)` provide the fitted layout. For simple univariate intercept blocks, the matrix axis name is `"(Intercept)"`. For multivariate intercept blocks, the axes have the form `trait:(Intercept)`. Known covariance between group levels is supplied through `ucov` separately from these coefficient covariance matrices.

Covariance matrices must be finite, symmetric, positive semidefinite, and consistent with the declared covariance family. `st.lmm.simulate` validates a fully specified model without evaluating a likelihood or solving for fitted coefficients. Zero variances and singular positive semidefinite covariances are therefore allowed, including deterministic responses when the complete covariance is zero. Fitted simulation also bypasses the observed data solve when `newparams$fixed` supplies a

known mean and no random coefficients are held. Otherwise, supplied covariance parameters require a positive definite observed covariance: omitted fixed coefficients are estimated by GLS, and held random coefficients are recomputed as BLUPs at the supplied parameters. Supplying known fixed coefficients recomputes these BLUPs relative to that mean. No covariance optimizer is used.

All retained response cells are simulated, including cells whose original responses were missing. Removed rows from an explicit `na.omit` or `na.exclude` fit are not reinserted; `newdata` specifies a different target layout. The missing-response mask is returned so a subsequent refit or bootstrap can reinstate the original observation pattern. Simulation needs the full covariance of the requested response cells, so its memory use grows quadratically with their number even when the fitted model used a sparse solver. New residual strata without fitted covariances, and new targets for direct observation kernels without a target-kernel definition, are rejected.

Value

A list of `nsim` matrices named `sim_1`, `sim_2`, and so on. Each matrix has one row per observation and one named column per trait, including models with a single trait. For a simulation from a specification with no response name, the column is named `response`. Row names follow the target data. Attributes include `seed`, the actual initial R random number state (this records `.Random.seed`, not any generator state external to it); `missing.response`, the original response-cell mask in the same matrix layout; `random.held`, a named indication of blocks with any held coefficients; `random.held.coefficients`, the exact named coefficient masks; `fixed.known`, the fixed-parameter interpretation; and `conditioning`, a description of the simulation convention. `input.rows` identifies retained original training rows when available.

See Also

`st.lmm`, `st.lmm.components`, `st.lmm.refit`, `simulate`.

`st.load.performance.data`

Loads 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")

st.generate.phenotypes(data.set, values, h2 = 0.8, trait = "y")
```

Arguments

performance.data	A data frame or matrix containing individual identifiers and numeric performance values.
data.set	Name of the destination marker or supported hybrid dataset.
values	Finite numeric vector in dataset order, or a prediction data frame returned by st.predict. For a data frame, identifiers must match the target rows exactly.
h2	Heritability, greater than zero and at most one.
trait	Phenotype column for a hybrid dataset. For an ordinary marker dataset the stored response is y. A parental identifier column cannot be replaced.

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 be non-empty, subject to the R string-size limit. Whitespace and control characters are rejected; ordinary punctuation such as dots, underscores, and hyphens is permitted. 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 may be finite numeric values or NA. An explicit NA record retains that individual and its marker genotypes, with a missing value in the stored performance vector. This differs from an individual without any matching performance row, which is removed. The character token "NA" is also accepted; other nonnumeric values, NaN and infinities are rejected. The resulting marker data set retains the current marker names and linkage map, subsets both allele arrays to individuals with a matching record, 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.

Phenotype generation from genetic predictions

Adds independent normal residuals to supplied genetic predictions and stores the resulting phenotype in a marker or supported hybrid dataset.

With sample variance s_g^2 of the supplied predictions, residual variance is $s_e^2 = s_g^2(1 - h^2)/h^2$. Finite-sample realized heritability need not equal the specified value. At least two records are required; predictions must vary when h2 < 1. Uses the R random-number generator. Dataset identifiers and pedigree are preserved. Replacing the response invalidates affected active fitted state. This operation does not send values to simulation populations.

The generated numeric phenotype vector, invisibly. The destination response is updated.

See Also

st.read.performance.data for reading performance data from a local file.

Examples

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
data("v-tropmaize-phe", package = "SelectionTools")
performance <- v.tropmaize.phe[match(vcf$individual,
                                     as.character(v.tropmaize.phe$ii)), ]
names(performance) <- c("individual", "performance")
st.load.performance.data(performance, data.set = "maize.example")
```

---

st.load.vcf.data *Loads Compact STvcf Marker Data with an Optional Linkage Map*

Description

Loads a compact STvcf object directly into a SelectionTools marker data set and, when present, its linkage map without intermediate marker or map text files.

Usage

```
st.load.vcf.data(STvcf, data.set="default", strictACTG=FALSE)
```

Arguments

STvcf	A list in STvcf format.
data.set	Character string naming the SelectionTools marker data set to be created or replaced.
strictACTG	Logical, default FALSE. When enabled, the reader accepts only A, C, G and T labels or the abbreviations Ade, Cyt, Gua and Thy, matched case-insensitively, and assign fixed internal codes A/Ade=1, C/Cyt=2, G/Gua=3 and T/Thy=4 independently of REF/ALT order. Other labels become missing.

Details

With strictACTG=TRUE, nucleotide labels determine the internal codes. Accepted labels are A, C, G, T, Ade, Cyt, Gua and Thy, case-insensitively. Markers having none of these labels in REF or ALT are discarded. A marker with an accepted label remains even if all its genotypes are missing. A diploid call containing any excluded allele becomes entirely missing. The surviving marker data and map remain aligned, and chromosome codes are consecutive after filtering. Marker-only input

remains marker-only. If no marker remains, loading returns a failure status without replacing an existing destination. The input STvcf object is not modified.

Strict mode ignores the optional `allele.codes` and `prediction.state` attributes because labels define a new coding; individual pedigree metadata is retained. For ordinary compact exchange of SelectionTools data, including objects with numeric REF/ALT labels, retain the default `strictACTG=FALSE`. It preserves the existing coding and fitted-state behavior described below.

Fixed nucleotide coding supports matching markers with different REF/ALT orders; it does not perform marker matching, strand complementation or indel normalization.

The optional `allele.codes` attribute is a list of integer coding vectors, named and ordered by marker ID. It records the internal allele codes corresponding to REF and successive ALT entries. STvcf exchange retains this attribute so fixed or subsetted loci keep the same coding as their fitted marker reference. The loader validates it before replacing a destination. Conversions preserve it; a manually edited marker list must keep it aligned. It is not part of plain-text VCF exchange.

The optional versioned `pedigree` attribute is a data frame with `name` and `pedigree` columns and integer attribute `version=1L`. Its names must match the individual identifiers in order. Without it, individuals are founders. Full strings are owned independently of ancestor populations. The optional `prediction.state` attribute preserves a compatible active symbolic-A fitted state during STvcf/RDS exchange. Conversion in either direction retains both attributes; loading validates them before replacing the destination.

The optional `physical.map` attribute is a data frame with `marker`, `chromosome` and `position` columns and `position.unit = "bp"`. Its marker identifiers must be unique and cover every imported marker; chromosome labels must be nonmissing and positions finite and nonnegative. The loader copies this map into compiled marker-data storage. It is independent of the genetic map in `CHROM/POS`; a physical map never supplies cM distances. Existing genetic-map inputs retain their previous interpretation. `st.gwas` uses the stored physical coordinates when `map = NULL`. Copying and subsetting marker data preserve the physical metadata, and STvcf export includes it again. Export of physical-map-only data retains missing genetic `CHROM/POS` and empty `CHROM.levels`.

STvcf is a compact exchange representation for diploid marker data with VCF-style allele indexing and optional genetic positions. 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, or all `NA_integer_` for marker-only data.

`POS` Double-precision genetic positions in cM. Values must be finite and nonnegative for mapped data, or all `NA_real_` for marker-only data.

`ID` Unique non-empty marker identifiers containing fewer than 256 bytes. Whitespace and control characters are not permitted.

`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 non-empty individual identifiers, subject to the R string-size limit. Whitespace and control characters are not permitted.

`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. An empty character vector declares marker-only data.

Individual names must be unique and non-empty, subject to the R string-size limit. Whitespace and control characters are rejected; ordinary punctuation such as dots, underscores, and hyphens is permitted.

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.

The input object can be prepared with `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. `"."` denotes `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. Fixed loci are accepted by the symbolic-A relationship path and contribute zero; other methods retain their own marker requirements.

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.

Mapped input must provide genetic map positions in centimorgans for all markers. 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. `st.get.map` reports the map positions used by SelectionTools.

For marker-only input without a linkage map, every `CHROM` entry is `NA_integer_`, every `POS` entry is `NA_real_`, and `CHROM.levels` is `character()`. All nine components remain required. Partial maps, nonmissing positions without chromosome levels, and incomplete chromosome assignments are rejected. Marker-only loading preserves input marker order and installs no map. It supports marker statistics and relationship-matrix construction; map-dependent operations need a genuine linkage map. Further details appear in `v-flintmaize-vcf` for supplied marker-only data.

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

An integer status, invisibly: zero on success and a negative value on rejected input. The supplied STvcf marker and map information is installed in `data.set` only after successful staging.

See Also

`st.dataframe.to.STvcf`, `st.markerdata.to.STvcf`, `st.STvcf.to.dataframe`.

Examples

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
st.get.map(data.set = "maize.example")
```

`st.lsm` *Least-squares means, contrasts and fixed-effect hypotheses*

Description

Constructs reference grids and estimates marginal linear functions of an `st.lmm` fit. Numerical construction and inference are implemented in C. The implementation follows the reference-grid methodology of **emmeans**; that package is needed only for optional reference comparisons.

Usage

```
st.lsm(object, specs, by = NULL, at = list(), weights = "equal",
       ddf = NULL, level = .95, adjust = "none", type = "link",
       transform = NULL, trend = NULL, delta.var = NULL,
       cov.reduce = mean, cov.keep = "2", rg.limit = 10000L,
       nesting = NULL, bias.adjust = FALSE, sigma = NULL,
       regrid = NULL, max.degree = 1L, L = NULL, rhs = 0,
       joint = FALSE, side = "two.sided", test = "F", DF = NULL,
       vcov. = NULL, mvt.control = NULL, calpha = NULL,
       method = NULL, ref = NULL, wts = NULL, scores = NULL,
       reverse = FALSE, infer = NULL, inference.space = "broad")
st.lsm.cld(object, alpha = .05, adjust = "tukey",
           sort = TRUE, reversed = FALSE)
st.lsm.test(object, joint = TRUE, by = NULL, rhs = 0, ...)
st.lsm.mvcontrast(object, method = "eff", mult.name = "trait",
                  by = NULL, adjust = "sidak", rhs = 0,
                  show.ests = FALSE, ...)
## S3 method for class 'st.lsm.mvcontrast'
print(x, digits = NULL, top = NULL, bottom = NULL,
  char.per.line = NULL, tests = NULL, ...)
## S3 method for class 'st.lsm'
print(x, digits = NULL, top = NULL, bottom = NULL,
  char.per.line = NULL, tests = NULL, ...)
## S3 method for class 'summary.st.lsm'
print(x, digits = NULL, top = NULL, bottom = NULL,
  char.per.line = NULL, tests = NULL, ...)
## S3 method for class 'st.lsm'
summary(object, level = NULL, adjust = NULL,
                        type = NULL, rhs = 0, side = "two.sided",
                        mvt.control = NULL, bias.adjust = NULL,
```

```
                          sigma = NULL, DF = NULL, calpha = NULL, ...)
## S3 method for class 'st.lsm'
as.data.frame(x, ...)
## S3 method for class 'st.lsm'
coef(object, ..., rhs = FALSE)
## S3 method for class 'st.lsm'
vcov(object, ...)
## S3 method for class 'st.lsm'
confint(object, parm, level = .95, ...)
## S3 method for class 'st.lsm'
pairs(x, reverse = FALSE, ...)
```

Arguments

object, x	An st.lmm fit for st.lsm; an st.lsm object for the other functions. The multivariate contrast print method takes a st.lsm.mvcontrast result. An existing least-squares means or contrast object is also accepted by st.lsm with method to construct contrasts, with symbolic L to define hypotheses on its displayed means, or with joint=TRUE to test its linear functions.
specs	Predictor names, or a one-sided formula such as ~treatment, ~treatment * year, or ~treatment | year. ~1 requests the overall mean. This argument is absent when L or an existing st.lsm object is supplied; the forms are mutually exclusive.
by	Conditioning variables. Contrasts retain the object's by groups when NULL; character() combines them.
at	Named list of reference values for predictors.
inference.space	The population described by the least-squares means and their SEs. The default "broad" averages over the random effects. The value "narrow" describes the specific groups in the experiment for all fitted random terms. A one-sided random formula or a character vector of fitted random block names selects which groups are specific to the experiment in an intermediate space. The value ~0 also gives broad inference. Intermediate and narrow inference use reference-grid means from ML or REML fits.
weights	"equal", "proportional", "outer", "cells", "flat", or nonnegative numeric weights for the averaged grid cells.
ddf	NULL retains the fit's method; otherwise "Satterthwaite", "Kenward-Roger", or "asymptotic" for likelihood fits. Kenward–Roger requires REML. For a fit using method="moments", NULL retains the fit's choice (default "Satterthwaite"). Single-stratum functions retain exact DF; functions combining strata use Satterthwaite DF. An explicit "exact" requires exact inference.
level	Confidence level, strictly between zero and one. NULL in summary retains the object's value.
adjust	Multiplicity method: "none", "tukey", "scheffe", "bonferroni", "sidak", "holm", "hochberg", "BH" (or "fdr", "FDR"), "BY", "mvt", "single-step", "free", "Shaffer", "Westfall", or "hommel". Complete pairwise contrasts

default to Tukey. The new "dunnett", "trt.vs.ctrl1", "trt.vs.ctrlk" and "mean_chg" families default to mvt; "del.eff" defaults to FDR. Existing other families retain their default of none; an explicit adjust="mvt" requests adjusted "trt.vs.ctrl" inference.

type — "link" or "response". NULL retains the stored choice.

transform — Response transformation: NULL detects a direct log, log2, log10, log1p, or sqrt response; "identity" disables back-transformation. May be specified for a precomputed transformed response. Also accepts "inverse", "logit", "probit", "cauchit", "cloglog", or a named list with linkinv and mu.eta functions and optional name.

trend — Optional numeric predictor name. Estimates its local slope instead of the least-squares mean. Slopes remain on the linear predictor scale.

delta.var — Finite-difference increment for trends; default is 0.001 times the observed predictor range. Must be nonzero.

method — NULL constructs least-squares means from a fitted model. Otherwise constructs contrasts of an existing st.lsm object, or of the means specified by specs in the same call. Available choices are "pairwise", "revpairwise", "trt.vs.ctrl", "consec", "eff", "del.eff", "poly", "opoly", "helmert", "mean_chg", "identity", "tukey", "dunnett", "trt.vs.ctrl1", "trt.vs.ctrlk", a numeric matrix with one row per contrast, or a named list of custom coefficient vectors. Additional sample-size weighted families are "Dunnett", "Tukey", "Sequen", "AVE", "Changepoint", "Williams", "Marcus", "McDermott", "UmbrellaWilliams" and "GrandMean". These capitalized names use **multcomp** contrast conventions and default to "single-step"; existing lowercase families retain their defaults, scaling and ordering. Their wts option supplies positive sample-size weights, defaulting to equal weights. Each vector has one entry per mean in each by group, in displayed grid order.

ref — Control indices or exact displayed level labels, including a vector to compare treatments with the average of several controls. NULL selects the first mean, or the last for "trt.vs.ctrlk".

cov.reduce — Function, named list of functions, or logical. The default takes each numeric covariate's mean. Functions may return several reference values, for example range. FALSE retains all observed values. Formula-based covariate reduction is not implemented.

cov.keep — Names of numeric covariates to retain at their observed values, and/or a character or numeric threshold for retaining covariates with at most that many unique values (default "2"). Retained covariates are treated as factors when computing cell weights.

rg.limit — Positive maximum number of crossed reference-grid rows, default 10000. Larger grids require a higher limit; the covariance of many requested means can require substantial memory.

nesting — NULL detects nesting from observed memberships and hierarchical fixed terms. A named list such as list(village="region") supplies it explicitly; list() disables detection.

bias.adjust — Controls the second-order Taylor bias correction to response summaries or regridding. NULL in summary retains the stored choice.

`sigma` Standard deviation for bias adjustment, scalar or one per summary/reference-grid row. NULL uses the square root of observation-averaged residual variance for the corresponding trait. For heterogeneous residuals or a different target distribution, an explicit sigma specifies the intended target distribution.

`regrid` NULL or `"none"` retains the linear-predictor grid. `"response"` transforms each grid prediction before averaging, propagates delta-method covariance, and places subsequent contrasts/tests on the response scale.

`max.degree` For trends, integer 1–5, default 1. Degrees above one add a `degree` conditioning column. For polynomial contrast families, omitting this argument selects the smaller of six and the number of means per group minus one. An explicit value selects the contrast degree.

`mvt.control` Named integration controls: `maxpts=131072`, `abseps=1e-5`, `releps=0`. The compiled deterministic integrator preserves R's random-number state. Summaries expose `mvt.error` and `mvt.converged`; the printer reports when the requested tolerance was not reached.

`L` Optional numeric coefficient vector or matrix, with one column per fixed coefficient, in `names(coef(object))` order. Named columns are aligned by name. Rows need not sum to zero. Estimable row-wise hypotheses require positive standard errors. Also accepts character or expression vectors containing linear equalities or inequalities. Symbolic names must be coefficient names when `object` is an `st.lmm` fit. With an existing `st.lsm` object, symbolic names instead identify its displayed least-squares means or contrasts. One factor is identified by its displayed level names; several factors are identified by displayed values joined with single spaces in the order of the object's label columns, excluding `by` columns. Nonsyntactic names require backticks. Names must be unique within each by group. The same hypotheses are resolved separately within every group, preserving the object's averaging, weights, covariance and offsets. An unknown or ambiguous name is an error; no reference level is substituted. Numeric hypotheses on an existing least-squares means object use `method`, not `L`. Exactly one of `method` and `L` is supplied, with `specs` absent when supplying an existing least-squares means object. Oppositely directed inequalities are sign-reversed into one common family direction; the stored matrix records that orientation. Only affine arithmetic with constants, `+`, `-`, `*` and `/` is accepted; expressions are never evaluated as R commands.

`rhs` In summaries, the null value on the linear predictor scale; omission retains the stored right-hand side. For `coef`, TRUE extracts the stored null vector. Otherwise a finite scalar or one null value per row of `L`. Symbolic hypotheses specify their own right-hand sides, including nonzero values; this argument is absent for them. A symbolic equality can compare one mean with a constant, or any affine linear combination of means with another such combination. A zero-null test requires an explicit zero right-hand side.

`joint` For `st.lsm`, TRUE selects a model-based joint test, including full hypothesis-specific Satterthwaite or Kenward–Roger inference. This also applies to symbolic hypotheses on an existing least-squares means object, after composing them with its stored coefficient matrix. Joint tests require equalities. Inequalities use row-wise inference and cannot be mixed with equalities in the same family.

Regridded response-scale objects retain the existing restriction against model-based joint tests. For `st.lsm.test`, the existing least-squares means joint convention is retained for ordinary grid objects; direct coefficient objects use the model-based test. FALSE selects row-wise inference.

`test` `"F"` or `"Chisq"` for a direct joint hypothesis. The chi-squared test is an asymptotic Wald test with raw likelihood covariance, without Kenward–Roger scaling. Ordinary and grouped grid joint tests use F. A supplied `DF` is unavailable for a chi-squared test.

`DF` Optional positive scalar or `Inf` for a common Wald reference number of degrees of freedom. Direct coefficient construction with `DF` requires `L` and a likelihood fit and uses Wald covariance; `ddf` is absent or `"asymptotic"`. Summary can explicitly select a reference DF without changing the stored object. The contrast path also accepts this summary override. Grid construction without `method` requires direct `L` for a constructor DF. Ordinary joint grid tests accept `DF` for likelihood fits and use the Wald covariance. Moment joint tests retain their error strata. Supplied DF are distinct from model-derived Satterthwaite DF.

`vcov.` Optional fixed-coefficient covariance matrix or a function of the fitted object returning that matrix. Available with direct `L` and likelihood fits. Dimnames are aligned, and dimensions, symmetry, finiteness and positive semidefiniteness are checked. The matrix is not repaired. Inference uses a supplied `DF`, or the normal approximation if `DF` is omitted; model-derived finite-sample DF are not applied to an arbitrary supplied covariance. The fitted object is not modified.

`calpha` Optional positive numeric critical value for confidence limits. It overrides the interval critical value only, not the reported p-values.

`mult.name` One or more marginal-grid factors whose levels form the response vector for multivariate contrasts; default `"trait"`.

`show.ests` TRUE includes the individual contrast estimates. With FALSE, the estimates table has zero rows.

`alpha` Significance threshold for letter displays.

`sort` TRUE sorts letter-display rows by mean (the default). FALSE retains reference-grid row order; letter assignment still follows the direction selected by `reversed`.

`reversed` FALSE (default) assigns letter `a` starting with the smallest displayed mean; TRUE starts with the largest. Letter assignment is independent within each conditioning or trait group. When `sort=TRUE`, rows follow the same direction. Ties retain reference-grid order.

`wts` Optional weights for effect contrasts or positive sample-size weights for capitalized contrast families. One entry per mean in each conditioning group, in displayed order.

`scores` Optional distinct finite scores for polynomial contrasts, one per mean in each conditioning group.

`infer` For moment fits, TRUE requests intervals and tests, FALSE retains point estimates, or a logical pair selects intervals and tests separately. NULL retains the usual inference. Likelihood fits require ordinary inference.

`reverse` Reverse contrast directions; `pairs` reverses pairwise differences.

side — "two.sided" (or "="), ">", or "<". One-sided tests return matching one-sided confidence limits. Tukey and Scheffe support only two-sided inference.

tests — Printing option. NULL omits tests against zero for plain least-squares means and retains test columns for contrasts. TRUE includes available test columns; FALSE omits test statistics and p-values. Estimates, standard errors, degrees of freedom and confidence intervals remain available.

digits — Decimal places for printed estimates and standard errors (integer 1–16; default 4). Small nonzero values and p-values are shown using bounds, and t ratios exceeding 1000 in absolute value are bounded. Scientific notation is not used. Stored numbers retain full precision.

top, bottom — Nonnegative integer numbers of leading and trailing rows to display in each table (defaults 10 and 5). Tables with no more than their sum are shown in full. Otherwise an explicit omitted-row count separates the two parts. Either argument may be Inf to show every row; zero suppresses that part.

char.per.line — Maximum output line width (default 75; integer 40–10000). When full pairwise comparison labels exceed the available space, the printer shortens distinguishable factor levels and gives a key for the shortened levels. If the table still exceeds the width, the confidence limits appear in a separate block with the same comparison labels. The compiled printer first reduces spacing, then decimal places (to at least two, or one if explicitly requested), then abbreviates labels with a printed key. Probabilities and existing numerical bounds keep the requested precision. If needed, it splits wide tables into column blocks with repeated row numbers and the first column. Extremely wide numbers use wrapped row records. Prose and formulas also wrap. UTF-8 widths are measured conservatively in bytes without splitting a character. Printing does not change stored numbers or labels.

parm — Reserved; confidence intervals are returned for every row.

... — Optional settings forwarded by methods. Contrast options are named arguments of st.lsm; pairs forwards them to that path. Ordinary joint grid tests accept a denominator-DF method through ddf. For likelihood fits with finite model-derived DF, the ordinary grid test uses the minimum across its independent hypothesis basis. Regridded ordinary joint tests retain the minimum stored DF within each group unless supplied DF provide a Wald reference. For summary, additional arguments are accepted for compatibility with the generic and are not used.

Details

st.lsm(e, method="pairwise") compares an existing means object, or st.lsm(m, ~treatment, method="pairwise") to construct the means and contrasts in one call. A named list or numeric matrix in method acts on the displayed means within each conditioning group; L instead acts on fitted coefficients. The two forms are mutually exclusive. pairs(e) is shorthand for the pairwise contrast path. method must be explicit when constructing contrasts with st.lsm; it is omitted when requesting a joint test of an existing object. Combining method and joint=TRUE tests the newly constructed contrasts.

Omitted contrast settings inherit the source object's confidence level, scale, transformation, bias adjustment, sigma and DF method. The adjustment defaults to the selected contrast family's con-

vention, independently of the adjustment on the means. An explicit `adjust="none"` disables multiplicity adjustment. Existing grid construction defaults are unchanged. `st.lsm(e, method=...)` replaces calls to the former exported contrast constructor. All contrast options are named arguments of `st.lsm`.

Printed tables use the heading Least-squares means and the column labels `lsmeans`, `DF`, `F`, `t`, `p` and `group`. Joint tests use `DF1` and `DF2`. Stored result columns use the same statistical names. Compact-letter memberships are stored in `.group` and displayed as `group`. Compact-letter displays reserve a fixed display position for each letter, even across omitted rows. Very wide letter sets use consecutive group panels. Printed alignment does not change group memberships.

Predictor-label columns that share a name with a result column receive the suffix `.predictor`; a numeric suffix distinguishes further collisions. For example, a factor named `p` appears as `p.predictor`, while `p` contains test p values. Model formulas and reference-grid specifications retain the original predictor names.

The reference grid crosses observed factor levels and explicit `at` values. Numeric covariates with more than two observed values are set to their arithmetic means; numeric predictors with one or two values retain those values. Means use rows with usable model predictors and at least one observed response. For a multivariate fit, each trait is reported as a separate by family. The first requested predictor varies fastest. The grid size is checked against `rg.limit`. Unordered factors whose labels are all finite numbers are sorted numerically. Ordered factors and explicit `at` sequences retain their specified order.

Inference space and standard errors: The SE of a least-squares mean depends on the population it describes. A broad mean averages over the random effects. An intermediate or narrow mean includes predictions for specific groups in the experiment. The reference grid includes the predictors used by the selected random terms and uses groups present in the fitted data. The `at` argument can restrict those groups. The same grid weights apply to the fixed and random contributions, which are calculated from one fit.

The reported SE is the prediction-error SE at the fitted covariance parameters. It includes the covariance between the fixed estimation error and the random prediction error. For a selected random target, `ddf = "Satterthwaite"` approximates DF using the sensitivity of the target's prediction-error variance to the covariance parameters. `ddf = "asymptotic"` uses infinite DF. The default for a selected target is Satterthwaite unless the fit uses asymptotic inference. Contrasts of the resulting means carry the same random target and its prediction-error covariance.

Selected random effects apply to means on the linear predictor grid. They require fitted grouping levels and prediction recipes. Direct coefficient hypotheses, observation kernels, trends, response regridding, Kenward–Roger DF, joint tests and Scheffe, Shaffer and Westfall adjustments are outside the selected-target calculation.

Equal weights average all reference cells. Proportional weights use the joint frequencies of the averaged predictors; outer weights use the product of their marginal frequencies. Cells weights use observed cell counts within each requested mean, and flat weights average its occupied cells equally. Numeric weights follow the order of the averaged predictors in the fixed formula, with the first varying fastest. New fits save the fitted prediction recipes for `poly`, `scale`, `splines::ns` and `splines::bs`; these bases are reused at the reference grid. Formula expressions that produce factors, such as `factor(x)`, retain observed input levels. Older saved fits without basis metadata must be refitted or use precomputed columns. Raw matrix/list data columns still require separate named predictor columns.

Formula offsets and a scalar or data-derived `st.lmm(offset=)` argument are evaluated on the reference grid and added once to the estimated mean. Data columns referenced only by the argument offset are included in the grid and can be set through `at`. Trends include the offset's derivative and contrasts combine its grid values with their contrast coefficients. A training-only external offset vector has no unambiguous reference-grid value; `st.lsm` reports an error for that case. No arbitrary average offset is substituted.

Nested factors include their parents in the grid. Impossible nested combinations are excluded, and averages over nested factors are computed within parents. A nested conditioning variable also retains its parents for contrasts. Detection uses the fitted data, including when `at` narrows the parent levels. The detected relation is available as the object's `nesting` attribute.

Each reported estimate is a linear function of the fitted coefficients. Estimability is checked against the fitted design; non-estimable means and their standard errors are NA. Standard errors and denominator degrees of freedom are computed for each mean or contrast. Kenward–Roger uses adjusted covariance and hypothesis-specific degrees of freedom; For likelihood fits, Satterthwaite uses covariance derivatives and observed parameter information. No asymptotic fallback is substituted when finite-sample information is unavailable. Missing Satterthwaite derivatives can be computed from a previously fitted asymptotic model without refitting its covariance parameters.

Balanced mixed designs fitted with `method="moments"` support the same least-squares means interface for split-plot comparisons. For example, `pairs(st.lsm(m, ~Variety), adjust="none")` compares whole-plot treatments averaged over dates; `pairs(st.lsm(m, ~Date), adjust="none")` compares dates averaged over varieties; and `pairs(st.lsm(m, ~Date | Variety), adjust="none")` compares dates within each variety. Each final contrast selects its own error variance. Single-stratum contrasts retain the exact mean square and DF. `pairs(st.lsm(m, ~Variety | Date), adjust="none")` compares varieties within each date and combines the whole-plot and subplot errors.

By default, means and contrasts combining independent strata use Satterthwaite DF. If their estimated variance is $\widehat{V} = \sum_s a_s MS_s$, the denominator DF are

$$\nu = \frac{\widehat{V}^2}{\sum_s (a_s MS_s)^2/\nu_s}.$$

Here MS_s and ν_s are the observed stratum mean squares and their DF; the coefficients a_s are determined by the linear function and design. Covariances use the corresponding bilinear functions. No model refitting or likelihood estimation is needed. Nonpositive or numerically unresolved variances remain errors; negative raw variance components are not truncated.

`ddf="exact"` restricts inference to single-stratum functions. When an exact-only least-squares mean combines strata, `st.lsm` retains its point estimate and reference grid; contrasts of that grid can still have exact tests. Exact-only inference for a contrast combining strata raises an error. `infer=FALSE` requests point estimates explicitly.

For inference with balanced multivariate mixed moment fits, each mean or contrast involves a single trait. Its variance and DF use the corresponding diagonal entries of the stratum mean-product matrices. The off-diagonal entries contribute to covariance between estimates for different traits, as returned by `vcov`. Older saved fits without stratum geometry retain point estimates. A new fit supplies the geometry for inference.

Joint hypotheses use `st.lsm.test` on the contrast object. A common single stratum retains its exact F test. Otherwise, Satterthwaite joint tests combine the DF of covariance-orthogonalized one-dimensional hypotheses. Direct `st.lsm(..., L=L, joint=TRUE)` uses the same calculation.

Tukey adjustment requires balanced pairwise covariance and common DF within each family; with combined strata its reference is approximate. Bonferroni and Holm support contrast-specific DF.

Adjustments apply separately within by groups. Tukey applies to complete pairwise families; with two means it is evaluated as the exactly equivalent t test, including denominator DF below two. Scheffe uses the rank of the family's linear functions. Holm, Hochberg, BH and BY adjust p-values; their confidence intervals use Bonferroni and carry an `interval.adjust` attribute. Unsupported adjustment names report an error. Multivariate-t adjustment uses the actual correlation of each contrast family, including singular pairwise families. A common-factor quadrature handles Dunnett structures; other structures use conditional t probabilities and shifted, antithetic Halton integration. For agreement with **emmeans**, finite integration DF are rounded down after adding 0.25, with a minimum of one; DF above 9999 use the normal limit. Single comparisons retain their ordinary t inference. This interface supports only `st.lmm` Gaussian models and has no plotting interface.

Ordinary response summaries inverse-transform marginal linear predictions. Bias adjustment uses the same second-order Taylor estimand as **emmeans**, not an exact lognormal correction. `regrid="response"` averages transformed grid predictions instead. Without regridding, standard errors use the delta method, intervals are transformed endpoint-wise, and tests remain on the linear predictor scale. Log-scale differences with zero-sum coefficients become ratios. For square-root back-transformation, negative predictions and interval endpoints map to zero, with a zero derivative for negative predictions, following **emmeans**. Reciprocal intervals crossing zero are rejected. Trends use successive finite differences of the saved design basis. Higher-degree entries divide by the degree factorial, following `emmeans::emtrends`; they remain on the linear predictor scale.

For likelihood fits, joint tests use an orthonormal basis for the estimable hypothesis space, its full covariance, and the smallest denominator DF across that basis, following `emmeans::test(joint=TRUE)`. Dependent hypotheses require a zero null. The explicit new path `st.lsm(e, joint=TRUE)` instead selects the full model-based joint calculation. Existing calls retain the least-squares means convention. Unavailable DF remain unavailable. Regridding retains the minimum original grid DF for each average or contrast.

Multivariate mean contrasts use the covariance rank as numerator DF. With mean marginal DF v and rank r, Hotelling's statistic is converted to $F = T^2(v - r + 1)/(rv)$, with denominator DF $v - r + 1$, as in `emmeans::mvcontrast`. Adjustments are applied across these tests within the remaining by groups.

Letter-display significance thresholds do not change the stored confidence level for mean intervals. Mean intervals use the requested adjustment (Sidak when pairwise tests use Tukey).

Letter displays use insert-and-absorb on the adjusted pairwise tests. They require all comparisons to have available inference. Letters are named in increasing mean order by default and in decreasing mean order with `reversed=TRUE`, so the selected extreme receives `a`. Overlapping groups can require several letters on a mean, such as `ab`; more than one mean can share `a`. This naming changes neither the tests nor which pairs share a letter. Shared letters indicate failure to reject a difference, not evidence of equality. Letter symbols can differ from other implementations while encoding the same comparison graph.

Value

`st.lsm` returns a serializable S3 object. Objects for least-squares means and individual contrasts contain the linear-function matrix `L`, labels, covariance and inference settings. Objects for joint tests contain the test table in `joint.tests`. The table is empty for other `st.lsm` objects. This table is displayed by `print` and returned by `summary` and `as.data.frame`. For direct joint F tests, its

columns are DF1, DF2, F, F.scaling, p and method. For direct chi-squared tests, the columns are DF, Chisq, p and method.

st.lsm.test returns a test data frame. st.lsm.mvcontrast returns a list with estimates and tests data frames. Summary and confidence-interval methods for means and individual contrasts return a summary.st.lsm data frame with estimates, SE, DF, confidence limits, t ratios and p-values for the requested alternative. coef and vcov return linear predictor estimates and their covariance for means and individual contrasts. st.lsm.cld returns a summary.st.lsm data frame with an additional .group column.

Reported degrees-of-freedom method

For fixed-effects-only models fitted by moments, least-squares means and comparisons use the residual degrees of freedom and report exact. The inherited denominator-DF setting remains available in the object; the display describes the calculation used. Estimates, standard errors, confidence limits and p-values are unchanged by this reporting choice. For pairwise comparisons from mixed models fitted by moments, the heading reports exact when all comparisons have certified exact inference from the same error stratum and share its denominator degrees of freedom. Comparisons mixing strata retain their Satterthwaite heading. Supplied DF values do not trigger this automatic exact label. Individual least-squares means retain their own inference setting; an exact comparison does not imply exact inference for each separate mean.

Printing least-squares means

Pairwise-comparison headings identify the variables whose means are compared. Grouping variables remain identified in their existing group labels. Plain least-squares means omit the t-statistics and p-values for tests against zero from the printed display by default. Contrasts retain their test columns. Estimates, standard errors, degrees of freedom and confidence intervals remain displayed when available. Tables show each row's denominator-DF method when methods differ. Trend estimates identify their predictor, and grid results report explicit at values. Nonzero null values and one-sided alternatives are identified in test output. Back-transformed results identify the scale of the tests.

The footer identifies the p-value and confidence-interval adjustments used for the displayed results. Compact-letter displays report the comparison adjustment and the significance threshold for the letters separately. The confidence level of the means remains distinct from that threshold. The print option tests=TRUE includes available test columns; tests=FALSE omits them. The same option applies to printed summaries. These display choices leave the fitted object, summary data frames and as.data.frame results unchanged. For example, print(st.lsm(model, ~treatment), tests=TRUE) includes the tests. With tests=NULL, st.options()$print.tests.for.bare.lsmeans controls their display for bare least-squares means. Explicit tests values take precedence; contrasts retain tests by default.

Compact LSD and HSD summaries

Balanced categorical analyses can retain one or two common pairwise-difference thresholds. The named numeric vectors LSD and HSD identify these comparison strata, for example Within env and Across env. The compact thresholds table contains the same one or two summaries with comparison SE, DF, family size and alpha. No pairwise-comparison lists are retained for this feature. When available, the table is an attribute of the summary data frame. For a least-squares means result x, the call attr(summary(x), "thresholds") returns this table.

For confidence level $1 - \alpha$, comparison standard error SE and denominator degrees of freedom DF,

$$LSD = t_{1-\alpha/2,DF} SE$$

and, for a family of k requested means,

$$HSD = q_{1-\alpha;k,DF} SE/\sqrt{2}.$$

The studentized-range quantile is denoted by q. HSD equals LSD for two means. The SE and DF belong to the difference, rather than either individual mean. Families follow the conditioning factors and are separate for different traits. A within/across label identifies a common comparison stratum; it does not change the requested family size.

The summaries require complete, equally replicated categorical layouts, equal replication of grouped random effects and at most two identifiable common SE/DF/family-size combinations. Unbalanced data, more than two combinations, unavailable quantiles or comparison strata without an unambiguous factor label produce no stored LSD/HSD summaries and no footer. Continuous-predictor models, point-only requests and contrast objects do not acquire these summaries. Mixed-model thresholds retain the requested comparison-DF approximation. The scale is the analysis scale; response regridding uses its corresponding scale. Back-transformed presentation alone does not transform additive thresholds.

Without displayed tests, printing uses at most two lines, for example `Within env: LSD=12.3 HSD=15 (alpha=0.05)`. There is no separate heading or comparison table. The complete footer is omitted if its lines cannot fit the configured line width. The options `st.options(list(print.LSD=FALSE))` and `st.options(list(print.HSD=FALSE))` independently suppress the respective values; both default to `TRUE`. The compact values remain stored when printing is disabled. Summary confidence-level/DF overrides produce their corresponding summary values without changing the source object. Existing printed-test and compact-letter-display conventions are preserved.

Direct and joint coefficient hypotheses

Direct matrices work with ML and REML fits and with balanced orthogonal method-of-moments designs that provide coefficient inference. For moment fits with explicit pooling, coefficient inference continues to use the original unpooled fitted model and its independent error strata; pooled hypothesis ANOVA tables are stored separately. With `joint=FALSE`, the result is an `st.lsm` object: estimates remain $L\hat{\beta}$; stored `rhs` and `side` define its default tests. Direct summaries display the tested null alongside each estimate. Explicit summary arguments override them. With `joint=TRUE`, `st.lsm` returns an object whose stored `joint.tests` table is displayed by `print` and returned by `summary` and `as.data.frame`. For joint tests, consistent redundant rows are reduced to an independent subset; inconsistent right-hand sides and non-estimable hypotheses are rejected. Rank reduction preserves the original independent rows. Kenward–Roger requires REML and uses adjusted covariance, denominator DF and hypothesis-specific F scaling. Satterthwaite combines the moments of orthogonalized one-dimensional tests. Existing fit information is reused; missing covariance derivatives can be reconstructed when needed. No alternative DF method is silently substituted. For multivariate mixed moment fits, direct coefficient rows each concern one trait. A joint F hypothesis uses rows from a single trait. `st.lsm.test(e, by="trait")` produces separate joint tests for the traits in a reference grid or contrast object. Fixed-only multivariate moment fits provide trait-wise row inference and model-term MANOVA tests.

Simultaneous inference

`"single-step"` uses the joint normal or t distribution and one common reference DF per family: a supplied `DF`, otherwise the minimum marginal DF, without rounding. Direct-matrix `"mvt"` uses the same policy. Unequal Satterthwaite DF do not define an exact multivariate t law; this common-DF treatment is an explicitly reported approximation. Existing grid-based `"mvt"` calls retain their previous DF convention. `"free"` performs a correlation-based step-down adjustment. `"Shaffer"` and `"Westfall"` use logical closure of homogeneous, two-sided equalities, with a maximum of 20 hypotheses per family. Nonzero nulls, offsets and one-sided logical-closure requests are rejected. `"hommel"` completes the ordinary `p.adjust` method family. Holm, Hochberg, Hommel, BH, BY and the step-down procedures do not imply matching rectangular confidence intervals: the reported intervals use Bonferroni, identified in `interval.adjust`. A supplied `calpha` is identified separately. Integration controls and convergence information are retained; the deterministic compiled integration does not change R's RNG.

References

McLean, R. A., Sanders, W. L. and Stroup, W. W. (1991). A unified approach to mixed linear models. The American Statistician, 45, 54–64. doi:10.1080/00031305.1991.10475767.

Hothorn, T., Bretz, F. and Westfall, P. (2008). Simultaneous inference in general parametric models. Biometrical Journal, 50, 346–363. doi:10.1002/bimj.200810425.

Lenth, R. V. (2016). Least-squares means: the R package lsmeans. Journal of Statistical Software, 69(1), 1–33. doi:10.18637/jss.v069.i01.

See Also

`st.lmm`, `st.lsm-hypotheses`

`st.lsm-hypotheses` *Fixed-Effect Hypotheses Through st.lsm*

Description

The direct coefficient path in `st.lsm` tests $L\beta = \mathrm{rhs}$ for supported moment, ML and REML fits.

Details

`L` accepts a vector or matrix; `joint=TRUE` requests a joint F test. Named columns align with the fixed coefficients. A scalar or per-row `rhs` specifies nonzero null values. Consistent redundant rows are reduced; non-estimable or inconsistent hypotheses are rejected. Kenward–Roger uses adjusted REML covariance, hypothesis-specific denominator DF and F scaling. Satterthwaite combines orthogonalized one-dimensional moment approximations. `ddf="asymptotic"` selects a Wald F test with infinite denominator DF; `test="Chisq"` selects the global asymptotic Wald chi-squared test. Without `joint=TRUE`, the result stores the linear estimates and covariance and supports ordinary summaries and simultaneous inference. Symbolic linear equalities and inequalities are also accepted in `L`. The existing least-squares mean construction is unchanged; an existing means or contrast object can be passed to `st.lsm(e, joint=TRUE)` for the full model-based joint calculation.

Value

Both direct joint tests and row-wise construction return an `st.lsm` object. For a direct joint F test, `summary` returns the stored table with `DF1`, `DF2`, `F`, `F.scaling`, `p` and `method`.

See Also

`st.lsm`, `st.lmm`

---

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

Summarizes 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. The statistics are returned directly from memory.
ind	Logical. TRUE requests the individual missingness table.
mar	Logical. TRUE requests the marker statistics table.
gen	Logical. TRUE requests the genotype matrix table.
auxfiles	Logical. If TRUE, the result is returned directly from memory.

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

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
statistics <- st.marker.data.statistics(data.set = "maize.example")
statistics$genotypes
```

---

st.markerdata.to.STvcf

Converts 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

A stored physical map is retained in the exported `physical.map` attribute. A dataset with a physical map but no genetic map is exported with missing `CHROM/POS` and empty `CHROM.levels`; base-pair positions are never substituted for genetic positions in cM.

The optional `allele.codes` attribute is a list of integer coding vectors, named and ordered by marker ID. It records the internal allele codes corresponding to REF and successive ALT entries. STvcf exchange retains this attribute so fixed or subsetted loci keep the same coding as their fitted marker reference. The loader validates it before replacing a destination. Conversions preserve it; a manually edited marker list must keep it aligned. It is not part of plain-text VCF exchange.

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 either a usable genetic linkage map or a stored physical map. When a genetic map is present, every marker-matrix column must have exactly one active genetic 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 and non-empty. Marker names must contain fewer than 256 bytes; individual names use owned dynamic strings subject to the R string-size limit. Whitespace and control characters are rejected; ordinary punctuation such as dots, underscores, and hyphens is permitted. Names are never silently modified during this conversion.

Simulation exports retain each individual's stable name and complete stored pedigree. Imported founders keep their imported names. New breeding individuals use their creation population and serial number followed by the stored pedigree expression. Copying or renaming a population does not regenerate these identifiers.

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 compiled 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.

The optional pedigree attribute contains the versioned name/pedigree data frame. When an active symbolic-A fit is stored, prediction.state retains its versioned fitted snapshot and source role for STvcf/RDS persistence. These attributes leave the nine-component STvcf layout unchanged. Standalone observed responses, earlier effects and selection values, and simulation GValue/PValue arrays are not exported as dataset fields. The fitted snapshot contains the response and effect information needed for prediction; refitting after import requires loading the desired response into the marker dataset.

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.ocs *Optimum Parental Contributions*

Description

Finds parental gene contributions that maximize supplied additive merit under an upper limit on group relatedness and lower and upper contribution bounds.

Usage

```
st.ocs(merit, K, theta, lower = 0, upper = 1, control = list())

## S3 method for class 'st.ocs'
print(x, digits = NULL, top = NULL, bottom = NULL,
                     char.per.line = NULL, ...)
```

Arguments

merit	Finite numeric vector of parental additive merits. Each entry corresponds to the same position in both dimensions of K.
K	Finite symmetric positive-semidefinite relationship matrix on the scale used by theta. For an additive relationship matrix A, the kinship convention is K = A/2. Singular matrices are supported. Matrix names are either absent or supplied for both dimensions and equal. If merit is also named, its names must match these names in the same order.
theta	Finite upper limit on $c'Kc$, where c is the vector of parental gene contributions. The meaning and reference population of this limit follow from the supplied matrix.
lower, upper	Lower and upper contribution bounds, each supplied as an unnamed numeric scalar or as a vector of the same length as merit. Bounds satisfy 0 <= lower <= upper <= 1. Full-length named bounds follow the parental order.

`control` Named list of numerical controls. `tolerance` (default `1e-8`) controls numerical convergence; `maxit` (default `10000`) limits inner optimization iterations; `max.outer` (default `100`) limits iterations per outer search phase; `rank.tolerance` (default `1e-10`) controls numerical rank decisions. The method scales the merit vector and matrix internally; convergence tolerances apply on those scales. Diagnostic residuals and objective bounds are returned in the original input units. Unknown controls produce an error.

`x` An optimum-contribution result.

`digits`, `top`, `bottom`, `char.per.line` Optional printing controls. Unspecified values follow the corresponding `st.options` settings.

`...` Additional printing arguments.

Details

The optimization problem is

$$\underset{c}{\text{maximize}} \quad g'c \quad \text{subject to} \quad \mathbf{1}'c = 1, \quad l \le c \le u, \quad c'Kc \le \theta,$$

where g is `merit`. The returned `gain` is the weighted mean of the supplied merits.

Feasibility is checked using the contribution bounds and the minimum attainable relatedness. An infeasible request is reported explicitly; constraints are not relaxed. The numerical method also handles tied merits, active bounds and singular positive-semidefinite matrices. An iteration limit has a separate status from a converged solution.

The matrix scale remains fixed across solutions. A centered genomic matrix defines a relatedness index relative to its allele-frequency reference. Dividing an additive relationship matrix by two does not by itself turn this index into an absolute probability of identity by descent. Negative off-diagonal entries can occur in a centered genomic matrix.

The contributions are nominal parental gene shares before within-family selection. They do not specify parent pairs, integer family sizes or realized ancestry after selection. In a non-selfed crossing plan, a contribution above one half cannot be realized; even contributions bounded by one half may need additional pair and count feasibility checks. Such mating restrictions are separate from the contribution problem solved here.

Input vectors and matrices and the caller's thread settings are preserved. Calculations use the supplied numerical precision without rounding results through formatted text. Numerical tolerances and certificates describe floating-point convergence.

Value

A list of class `st.ocs` with the following components. Its printed representation summarizes the fit and the largest and smallest parental contributions under the current printing options.

`contributions` Named numeric vector in parental order, or `NULL` when the request is infeasible or no feasible incumbent has been found within the numerical limits.

`gain` Weighted parental merit, $g'c$, for the returned solution.

`relatedness` Achieved group relatedness, $c'Kc$.

`minimum.relatedness`
: Relatedness from the minimum-relatedness search under the contribution bounds. The difference from `diagnostics$minq.lower.bound` measures the remaining numerical gap in that search. The main status describes the merit optimization; a certified merit optimum can be available before this separate gap has converged.

`theta`
: Requested relatedness limit.

`status`
: `"optimal"` for a certified ordinary solution, `"minimum.relatedness"` for a certified boundary solution, `"infeasible"` for an infeasible request, or `"maxit"` when the iteration or numerical-certification limits prevent convergence.

`diagnostics`
: Numerical certificates, constraint residuals, objective bounds, iteration counts and matrix-rank information. A minimum-relatedness boundary may have a separate certificate because a finite relatedness multiplier need not exist there.

`bounds`
: Expanded lower and upper contribution bounds.

`control`
: Numerical control values used for the fit.

References

Wellmann R (2019) Optimum contribution selection for animal breeding and conservation: the R package optiSel. BMC Bioinformatics 20:25.

See Also

`gs.esteff.rr`, `gs.estimate.gv`, `st.get.A`, `gs.cross.info`

---

st.od *Constructs a path in the SelectionTools output directory*

Description

Constructs 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.oma *OMA: Optimum Mate Allocation*

Description

Chooses continuous allocations to allowed diploid matings to maximize their weighted merit under parental and mating bounds and two-generation relatedness constraints. OMA denotes optimum mate allocation.

Usage

```
st.oma(parents, matings, K, dF, base = "current", control = list())

## S3 method for class 'st.oma'
print(x, digits = NULL, top = NULL, bottom = NULL,
                    char.per.line = NULL, ...)
```

Arguments

parents	Data frame containing parental IDs and lower and upper contribution bounds. Its three columns are `id`, `min`, and `max`, in that order. IDs are unique, complete character strings. Bounds are finite and satisfy `0 <= min <= max <= 1`.
matings	Data frame containing allowed pairs, their fixed merit, and lower and upper allocation bounds. Its five columns are `parent1`, `parent2`, `merit`, `min`, and `max`, in that order. Parent IDs match `parents` and `K`. Self matings are supported. Each unordered pair occurs once; reversed duplicate rows produce an error. Merits are finite numeric values; bounds satisfy `0 <= min <= max <= 1`.
K	Finite symmetric positive-semidefinite diploid kinship matrix. Both dimensions have the same unique, complete parental names in the same order. It may contain additional reference individuals that do not occur in the allowed matings.
dF	Finite numeric vector of length one or two specifying a scalar change or lower and upper changes relative to the reference value. Both endpoints are strictly less than one; negative endpoints are supported. Two values are ordered from lower to upper. One value supplies both endpoints, giving an equality for the first-generation constraint.
base	Either `"current"` or `"RM"`. The former uses average current inbreeding as the reference; the latter uses a randomly mated reference including selfing. The supplied matrix and reference remain fixed throughout optimization.
control	Named list of numerical controls. Unknown and repeated names produce errors. The supported components are: `maxit` Positive whole-number iteration limit no larger than `.Machine$integer.max`; default `500`. `tolerance` Finite numerical tolerance for the reported scaled primal residual and normalized merit-bound gap, strictly between zero and one; default `1e-8`.

x	An optimum mate-allocation result.
digits, top, bottom, char.per.line	Optional printing controls. Unspecified values follow the corresponding st.options settings.
...	Additional printing arguments.

Details

For mating $e = (i, j)$, the continuous allocation x_e is the fraction of produced progeny assigned to that mating. Allocations sum to one. A parent's gene contribution is

$$c_i = \frac{1}{2} \sum_e x_e \left[\mathbf{1}(i_e = i) + \mathbf{1}(j_e = i) \right].$$

A self mating contributes both parental slots to the same parent. The objective is the allocation-weighted merit $h'x$. A supplied merit can represent a predicted cross mean or a fixed usefulness score. Score calculation and the biological selection scenario are determined before allocation; changing allocation alone does not recalculate the score. A subsequent resource allocation with different family sizes requires scores for those sizes. Contribution constraints concern nominal gene shares before within-family selection, rather than realized selected-descendant ancestry.

For diploids, the first- and second-generation quantities are

$$F_1 = \sum_e K_{i_e j_e} x_e, \qquad F_2 = c' K c.$$

With base = "current", the reference is $F_0 = \text{mean}(2\,\text{diag}(K) - 1)$. With base = "RM", it is $F_0 = \text{mean}(K)$. These means use the complete supplied matrix before restriction to eligible mating parents. Extra reference individuals can therefore change the constraints even when they do not occur in any selected mating.

For dF = c(dL, dU), the constraints are

$$d_L + (1 - d_L)F_0 \le F_1 \le d_U + (1 - d_U)F_0,$$

$$F_2 \le d_U(2 - d_U) + (1 - d_U)^2 F_0.$$

The reported transformed quantities are $dF_1 = (F_1 - F_0)/(1 - F_0)$ and $dF_2 = 1 - \sqrt{(1 - F_2)/(1 - F_0)}$ where these are defined. A scalar dF is an equality for F_1, rather than an upper bound alone. The current reference is undefined when $F_0 = 1$; the randomly mated reference is the relevant alternative for fully inbred lines.

A centered genomic matrix describes a relatedness index relative to its allele-frequency reference. The formulas do not by themselves make that index an absolute identity-by-descent probability or an observed biological inbreeding rate. The first-generation average within-mating quantity and second-generation group quantity are distinct constraints.

Continuous mate allocation does not supply a fixed number of distinct families or integer progeny counts. Optimum cross selection with equal positive integer family sizes is provided by st.cross.plan. Conversion of continuous allocations into an operational integer plan is a separate constrained problem; allocations are not rounded automatically.

Value

An object of class `st.oma` with the following components:

allocations Numeric vector of continuous allocation fractions in the input mating order. The values sum to one for a feasible result. No output thresholding or integer rounding is applied.

contributions Named vector of parental gene contributions in the input parental order.

parents The supplied parental table with its contribution column.

matings The supplied mating table with allocation and kinship columns. All candidate rows are retained, including zero allocations.

F0, F1, F2 The full-reference value, allocation-weighted mating kinship and parental group quantity, respectively.

dF1, dF2 The corresponding transformed first- and second-generation changes, where defined.

merit The allocation-weighted supplied cross merit.

status `"optimal"` when the reported scaled primal residual and normalized merit-bound gap satisfy the requested tolerance. The gap is divided by the largest absolute deviation of supplied merits from the midpoint of their minimum and maximum, with a divisor of one for constant merits. The reported merit and gap retain their original units. Other statuses are `"infeasible"` when infeasibility is established, or `"maxit"` when the iteration or numerical limit leaves the request unresolved. A `"maxit"` result can retain a feasible incumbent. Without a feasible incumbent, allocations, contributions and their derived values are missing rather than a feasible mating plan.

diagnostics Numerical reason, iteration count, constraint residuals, infeasibility lower bound, matrix eigenvalue and curvature information, reference size and problem dimensions. The overall primal residual combines absolute allocation/simplex/contribution residuals with scaled first- and second-generation violations. Individual `F1.residual` and `F2.residual` values use the original kinship units. The `affine.roundoff` component records numerical uncertainty in the scaled first-generation equality. Components `last.candidate.residual`, `last.candidate.F1.residual`, and `last.candidate.F2.residual` describe the latest candidate assessed, which need not be a feasible returned incumbent.

bounds The requested two-element `dF` interval, transformed `F1.lower`, `F1.upper` and `F2.upper`, merit upper bound, remaining gap and bound description. The gap is evaluated using centered merits before the common merit offset is restored. This preserves the same simplex objective while avoiding cancellation under a large common offset. The original-unit `merit.upper` remains rounded upward; subtracting the rounded reported merit from it can therefore give a larger difference than the reported gap.

control The iteration limit and tolerance used.

inputs The supplied parent and mating tables, complete matrix, `dF` request and reference convention retained for reconstruction.

The print method returns `x` invisibly.

References

Endelman JB (2025) Genomic prediction of heterosis, inbreeding control, and mate allocation in outbred diploid and tetraploid populations. Genetics 229:iyae193. doi:10.1093/genetics/iyae193.

See Also

st.ocs, st.cross.plan, gs.cross.info, st.options

---

st.outdir *Constructs a path in the SelectionTools output directory*

Description

Constructs 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. Further details appear in 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. `FALSE` preserves missing target genotypes. `TRUE` retains inferred hard calls at missing genotypes of existing target markers. Observed unordered genotypes are preserved under both policies. Default: `FALSE`.

`commit`
: Logical scalar. `TRUE` writes the completed result to `data.set` and marks it phased. `FALSE` performs and validates phasing, discards the proposed genotypes, and returns only diagnostics 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 STvcf converter requires non-empty marker and individual names without whitespace or control characters; ordinary punctuation is permitted. 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; automatic windows or a larger explicit window can satisfy the conditions.

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 reported with `commit = FALSE`.

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. Coding compatibility is required 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 retained in the validated proposal 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 allele storage is changed. Original internal allele codes, map, phenotypes and fitted effects are retained. Marker statistics are refreshed with the committed genotypes; calculated design and relationship matrices are invalidated. A failed phasing or validation call does not partially write a chromosome into the source. Loaded marker data are read directly in C into chromosome workspaces. No complete STvcf snapshot, R genotype matrix or phased genotype object is created as an internal interface or returned. References are never written. As with other genotype-changing operations, previously computed analyses should be recomputed when their inputs have changed through imputation. Storage is selected automatically from the largest observed allele count at any marker, using 4, 8 or 16 bits per diploid genotype plus dictionaries where required. Imputation can promote storage when it introduces additional alleles. The storage width is reported by `st.data.set.list()`.

Threads and reproducibility. `st.set.num.threads` and `st.get.num.threads` control and report the thread budget. 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

With `commit = TRUE`, invisibly returns `NULL` after updating the supported data set. With `commit = FALSE`, returns a small diagnostics list and discards the proposed genotypes. No genotype or full reference object is returned. The dry-run diagnostics contain:

`actual.seed` The actual seed used, including a seed drawn from R.

`windows`, `chromosome.windows` Total and per-chromosome window counts.

`maximum.worker.team` Maximum actual worker-team size.

`requested.threads` Requested SelectionTools thread count.

`filled.genotypes` Proposed number of filled genotypes.

`committed` Always `FALSE` for returned dry-run diagnostics.

`window.settings` Effective per-chromosome window choices.

`reference` Reference sample and haplotype counts, invariant-check status and ignored-marker count, or `NULL` in cohort mode.

Subsetting

A copy of the source can be restricted with `st.restrict.marker.data` before calling `st.phase()` when only a subset of chromosomes or individuals is wanted.

`st.plot.corr` *Plots correlation information from SelectionTools results*

Description

Plots 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` *Plots correlation information from SelectionTools results*

Description

Plots 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 *Plots 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 applies

only 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 *Plots 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.predict *Predict Random Effects and Combined Level Means*

Description

Predicts selected random effects alone or additive combinations with the fixed intercept in an `st.lmm` fit. A single unprefixed random-term name retains the level-mean interface. Independent random effects and supplied, possibly singular, relationship matrices are supported. Construction and numerical calculations are implemented in C.

Usage

```
st.predict(object, term = NULL, level = .95, newdata = NULL)
## S3 method for class 'st.predict'
print(x, digits = NULL, top = NULL, bottom = NULL,
                          char.per.line = NULL, ...)
## S3 method for class 'summary.st.predict'
print(x, digits = NULL, top = NULL,
                          bottom = NULL, char.per.line = NULL, tests = NULL, ...)
## S3 method for class 'st.predict'
summary(object, level = NULL, ...)
## S3 method for class 'st.predict'
as.data.frame(x, ...)
## S3 method for class 'st.predict'
coef(object, ...)
```

```
## S3 method for class 'st.predict'
vcov(object, ...)
## S3 method for class 'st.predict'
confint(object, parm, level = .95, ...)

st.marker.results(data.set = "default")
```

Arguments

object, x	An st.lmm fit or the name of a marker dataset containing stored symbolic-A prediction state for st.predict; a prediction object for its methods, or a prediction summary for its print method.
newdata	For marker prediction, the name of a target marker dataset or a supported target hybrid dataset. NULL with a marker-dataset object returns stored training conditional predictions.
term	NULL for marker-dataset prediction, otherwise a random-term name or an additive prediction expression, for example "gen" or "overlay(mother, father)". The selected term must be a random intercept with a constant unit basis, or a parental overlay.
level	Pointwise coverage level, strictly between zero and one. NULL in summary retains the stored value.
digits, top, bottom, char.per.line	The same compiled printing controls as st.lsm. Defaults show four decimal places, ten leading and five trailing rows, within 75 characters per line. Either row limit may be Inf to show all rows. Printed degrees of freedom use DF. Printing does not alter stored values.
tests	For prediction summaries, NULL preserves available test columns, TRUE includes them and FALSE omits them. Stored results are unchanged.
parm	Reserved; intervals are returned for all levels.
...	In print and as.data.frame, optional level for the summary; otherwise reserved.
data.set	Name of a marker or supported hybrid dataset used as a source of a symbolic genomic relationship matrix in st.lmm.

Details

Additive prediction expressions select scalar random-intercept terms and parent columns of a fitted shared overlay. For example, st.predict(fit, "0 + cross") returns SCA effects alone, whereas st.predict(fit, "male + female + cross") returns the intercept plus both shared parental GCA effects and SCA. The parent names resolve to the same overlay coefficient vector; they do not define separate effects. Expressions use simple column names separated by +; a leading 0 + omits the intercept. Duplicate terms, unknown levels, random slopes and ambiguous overlay parents are rejected. This interface currently uses observed combinations in the fitted data, collapses identical prediction targets and retains their first occurrence order. It does not construct unobserved crosses or average additional fixed effects. The returned vcov is the full joint prediction-error covariance of the selected linear targets, including fixed–random and random–random cross-covariances. Variance

parameters are held at their fitted estimates. These expressions currently require a univariate ML or REML model.

An overlay term may be prefixed with `0 +`, for example `st.predict(fit, "0 + overlay(female, male)")`, to return parental GCA effects alone. The unprefixed form returns intercept plus GCA. Prediction-error covariance for the zero-intercept target uses the random effect block only, including uncertainty due to estimation of fixed effects; it does not add the intercept target or its cross-covariance terms.

The same interface accepts a complete overlay interaction block name, for example `"0 + overlay(female, male):env"`. It returns every parental effect within every independent grouping level, together with the full joint prediction-error covariance in that displayed order. Group tuples and parents use explicit length-prefixed labels, so names containing colons cannot merge distinct targets. An `at()` overlay creates separate fitted blocks; their exact names, including the stratum suffix, identify the requested blocks. An unprefixed request adds only the fixed intercept, retaining the usual fixed-predictor reference; it does not add an environment fixed effect automatically. Conditional `predict` with the target cross rows provides complete environment-specific cross means. Unused parents supplied through the parental relationship matrix are retained in each grouping level.

For parental overlay predictions, printing identifies numbered parents as P1, P2, and so on and distinguishes GCA effects alone from intercept plus shared GCA. Stored parent identifiers, estimates, covariance and summary-column names are unchanged.

For a series fit with only fixed environment effects, the shared genotype term returns the equal average of fixed environment means plus its BLUP. The full prediction-error covariance includes estimation of that average. A separate `reliability` attribute (also shown in the summary) is the variance of the genetic BLUP divided by its prior variance; it excludes uncertainty in the average environment effect. These calculations condition on fitted covariance parameters. This initial series prediction requires an estimable intercept and fixed environment effects only.

When `object` names a marker dataset, prediction uses its stored symbolic-A fitted state and does not require a separately retained `st.lmm` object. `st.predict(fit, newdata="target")` uses the same marker-target route from the fit's snapshot. All participating sources must retain the same active fitted state for dataset-based prediction. Source changes that make this state incompatible are rejected; an existing fit retains its own snapshot. Removing a source dataset or compacting its storage does not invalidate that fit. Target datasets must still exist when requested. compiled marker datasets own their relationship matrices, effects and genotypic predictions in C memory; returned R objects are independent copies. Removing an R object does not release backend values.

Target markers are matched by identifier and encoded using the fitted source alleles, centering and relationship scale. Target frequencies do not replace the training reference. Fixed source loci contribute zero. The new marker-target route supports univariate models with an intercept-only fixed model and genomic random intercepts, including hybrid GCA and factorized SCA terms. All fitted markers are required; additional target markers are ignored. Marker order may differ. Extending the target panel therefore does not require refitting the source model: cross relationships still use the fitted marker subset and its training reference. It returns point predictions without prediction-error standard errors. Ordinary target output has `gen` and `prediction`; hybrid output retains the target performance columns and adds `prediction`. This mode does not modify target phenotypes. With other fixed predictors, their required target columns must be available.

The following details concern the fitted-object, named-term interface.

For an `overlay(mother, father)` term, the result contains one row per parent in the common parent basis. Unless prefixed with `0 +`, it returns the fixed intercept plus that parent's GCA, includ-

ing their joint prediction-error covariance. Conditional `predict` with parent columns in `newdata` provides cross predictions.

For each level g, the prediction is $\widehat{\mu} + \widehat{u}_g$. Other random terms are excluded from the prediction, although they remain in the fitted model and its uncertainty calculation. For example, `st.predict(m, "gen")` excludes block deviations. If the fixed formula contains other predictors, only its intercept coefficient is included: numeric predictors are at zero and factor contrasts at their reference setting. This is not an average over a fixed-effect reference grid. An independently estimable fixed intercept is required.

The first implementation supports univariate Gaussian fits and random-intercept or parental overlay terms. A selected random slope, random-regression block, kernel term or multivariate response reports an error. Predictions remain on the fitted response scale, including any response transformation; there is no automatic back-transformation. All fitted levels are returned, including additional levels supplied through `ucov`. Factor levels retain their specified order, except that unordered numeric labels are sorted numerically, as in `st.lsm`. Additional nonnumeric levels follow the observed factor levels.

Genomic prediction of genotypes without phenotypes requires those genotypes in the row and column names of the supplied relationship matrix. Their responses may all be `NA`; these responses do not contribute to fitting, but `st.predict` still returns the genotype predictions and their prediction-error standard errors. For a comparison of two-step and one-step genomic prediction requires masking every plot response of the same held-out genotypes before either training analysis. In the two-step analysis, the field model fits the remaining plots with fixed genotype effects and produces their BLUEs. Only those training BLUEs supply observed responses in a genomic model with `y ~ 1`, `random = ~gen` and `ucov = list(gen = A)`; the held-out means remain missing. In the one-step analysis, the genomic model fits the same masked plot data directly, retaining the block terms. BLUEs from a separate complete-data field model may be used as an evaluation reference, but must not supply the training responses. Genotype identifiers determine prediction alignment; prediction row order need not equal the data order.

Standard errors are square roots of prediction-error variances for the latent level means, including fixed-intercept uncertainty and fixed/random cross-covariance. They do not include the residual variance of a future observation. Writing $V = \mathrm{Var}(y)$, $G = \mathrm{Var}(u)$, $K = \mathrm{Cov}(y, u)$, $C = (X'V^{-1}X)^{-1}$ and L for the repeated intercept-selection row, the prediction-error covariance is

$$G - K'V^{-1}K + (L - K'V^{-1}X)C(L - K'V^{-1}X)'.$$

The computation uses an estimable fixed-effect basis and does not invert G; zero random variance and singular relationship matrices are therefore supported.

Intervals use the normal approximation (`DF = Inf`) with the fitted variance parameters held fixed. They are pointwise empirical BLUP prediction-error intervals for latent means; they do not adjust for estimation of variance components. No fixed-effect t tests or p-values are attached to these random-effect predictions. The fit's Satterthwaite or Kenward–Roger setting is not used for these intervals.

Value

`st.predict` returns a serializable `st.predict` object containing `term`, `levels`, `estimate`, `vcov`, `level` and `point.predictions`. `coef` returns a prediction vector and `vcov` returns a matrix.

For a named random term, the matrix contains the full prediction-error covariance and the point-prediction table has zero rows. The summary is a `summary.st.predict` data frame with a level-label column, `prediction`, `SE`, `DF`, `lower.CL` and `upper.CL`. The level column uses the term name, or `level` if that name collides with a numeric output column.

For a marker-target call, `summary` and `as.data.frame` return the point-prediction table with the target identifiers. A dataset call without a target gives the training conditional predictions in the table, with one column per trait. For multiple traits, `coef` stacks the columns of this table in order. For both marker routes the prediction-error covariance is a zero-by-zero matrix.

Stored marker prediction state

Returns the active fitted effects and prediction information stored by a successful symbolic-A fit.

A successful top-level fit with quoted symbolic `"A"` or `"kro(a,a)"` replaces the participating source datasets' previous state. Failed or nonconverged fits do not replace it. Two parental sources of a hybrid fit share one model identity. Numeric covariance matrices and fits without symbolic covariance requests do not establish this association. Stored state is separate from the older `gs.return.effects` result. This accessor returns a private copy.

A versioned list containing `model`, `sources`, marker-reference `panels`, `dataset`, source hybrid `performance` where applicable, `genotype.effects`, `marker.effects`, and training conditional `predictions`. Marker effects are available for supported single-pool genomic terms; SCA is retained through relationship scores and parental kernels. The model identity is an attribute.

See Also

`st.lmm`, `st.lsm`

Examples

```
data("v-lmm-rapeseed", package = "SelectionTools")
m <- st.lmm(hgt ~ env, ~geno, data = rapeseed,
            ddf = "asymptotic", random.test = "none")
p <- st.predict(m, "geno")
p
coef(p)
vcov(p)
st.predict(m, "0 + geno")
```

---

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. A value of 1 retains positions already in the desired map unit.
auxfiles	Logical. If TRUE, the matched and sorted map is returned directly from memory.
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 selected 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 *Loads 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.

See Also

st.read.map, st.read.marker.data.df, define.genome.

Examples

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
dosage <- t(matrix(as.integer(vcf$allele1) + as.integer(vcf$allele2),
                   nrow = length(vcf$ID)))
dosage[dosage > 2] <- NA_integer_
```

```
dimnames(dosage) <- list(vcf$individual, vcf$ID)
st.read.marker.data.df(dosage, format = "matrix", data.set = "mapped.example")
map <- data.frame(marker = vcf$ID, chrom = vcf$CHROM, pos = vcf$POS)
st.read.map.df(map, data.set = "mapped.example")
```

`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 tables are returned directly from memory.
`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 individual, marker, and genotype summaries are returned directly from memory.

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

Loads 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"`. A pipe indicates 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`.

See Also

`st.read.marker.data`, `st.load.vcf.data`, `st.marker.data.statistics`.

Examples

```
data("v-tropmaize-vcf", package = "SelectionTools")
marker.frame <- st.STvcf.to.dataframe(v.tropmaize.vcf)[1:12, c(1:6, 7:14)]
vcf <- st.dataframe.to.STvcf(marker.frame)
st.load.vcf.data(vcf, data.set = "maize.example")
dosage <- t(matrix(as.integer(vcf$allele1) + as.integer(vcf$allele2),
                   nrow = length(vcf$ID)))
dosage[dosage > 2] <- NA_integer_
dimnames(dosage) <- list(vcf$individual, vcf$ID)
st.read.marker.data.df(dosage, format = "matrix", data.set = "matrix.example")
index <- which(!is.na(dosage), arr.ind = TRUE)
marker.list <- data.frame(individual = rownames(dosage)[index[, 1]],
                          marker = colnames(dosage)[index[, 2]],
                          allele = c("1/1", "1/2", "2/2")[dosage[index] + 1L])
st.read.marker.data.df(marker.list, format = "list", data.set = "list.example")
st.marker.data.statistics(data.set = "list.example")$genotypes
```

st.read.performance.data *st.read.performance.data()*

Description

Reads phenotypic values for a marker data set and retains marker-data individuals with a matching phenotype record, including explicit missing values.

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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 or the token NA. An explicit NA record keeps the individual and its marker genotypes, with a missing phenotype; an individual without a matching record is removed. NaN, infinities and other nonnumeric tokens are rejected. 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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. A copy made before haplotyping preserves the original marker-level representation when both representations are 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; the rule is described in 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 *Resets SelectionTools data and settings*

Description

Resets SelectionTools data and settings.

Usage

```
st.reset()
```

Value

An invisible `list` returned by the compiled 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. This is mutually exclusive with ind.file.
ind.file	Optional file containing individual names to retain. Relative names are read from the input directory. This is mutually exclusive with ind.list.
mar.list	Optional character vector of marker names to retain. Unknown names are ignored. This is mutually exclusive with mar.file.
mar.file	Optional file containing marker names to retain. This is mutually exclusive 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. A prior copy with st.copy.marker.data preserves the original data set.

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. Separate calls provide staged filtering. For example, one call removes individuals with high missingness and a later call filters markers using the updated statistics.

The linkage map and associated performance data are subset 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

Returns performance data from a SelectionTools data set

Description

Returns 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. The result is returned directly from memory; this argument does not name an output file.
`auxfiles`	Logical. If `TRUE`, the result is returned directly from memory.
`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 *Selects individuals using phenotypic information*

Description

Selects 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 *Sets haplotype-block information*

Description

Sets 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. The result is returned directly from memory; this argument does not name an output file.
auxfiles	Logical. If TRUE, the result is returned directly from memory.
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 *Sets info level used by SelectionTools*

Description

Sets 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 compiled 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 *Sets matrix ops used by SelectionTools*

Description

Sets 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 *Sets the computational thread budget*

Description

Sets the thread budget for SelectionTools OpenMP calculations and, when its runtime interface is available, for the BLAS library used by R.

Usage

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

Arguments

active	One finite integer. Values below 1 are set to 1 and values above 256 are set to 256.

Details

The explicit setting is stored by SelectionTools and supplied to its parallel numerical kernels. It also sets the calling thread's OpenMP maximum and the configured thread count of a supported BLAS. OpenBLAS, FlexiBLAS and MKL interfaces are used only when their control routines can be identified in the library supplying R's linked BLAS operations. No BLAS library or R package is loaded to provide these interfaces. Other BLAS implementations remain usable, but their thread counts cannot be queried or changed by this function.

The package startup hook calls this function with a budget of 2. A subsequent call after loading the package selects another budget. Older routines allocate per-thread work for at most 256 threads, so the requested value is clamped to that range. With OpenMP, the effective package and BLAS budgets are also capped by the runtime thread limit. An explicit setting disables OpenMP dynamic team shrinking to keep the package and OpenMP BLAS budgets consistent. The setting applies to the current R process; other packages using the same OpenMP runtime or BLAS may also be affected.

Backend limits and the size of an operation can reduce the number of threads actually used. Builds without OpenMP perform package calculations serially. A detected pthread OpenBLAS can still use the requested BLAS budget. A build without OpenMP limits an OpenMP OpenBLAS, or FlexiBLAS with an unknown backend threading model, to one BLAS thread because it cannot synchronize the OpenMP runtime. This avoids waiting for BLAS workers that cannot start. The reference BLAS supplied with R is serial and has no thread-control interface.

The parallel kernels used by `st.lmm` do not call BLAS or the R API from their worker threads. The older genomic-selection cross-validation and permutation-test loops call BLAS and therefore execute their outer iterations serially. This avoids concurrent BLAS calls from OpenMP teams with backends for which that combination is unsafe. BLAS operations in these iterations retain the configured BLAS budget, so the operations themselves can still use multiple threads. This conservative policy can reduce parallel speedup of those older outer loops.

`st.get.num.threads(complete = TRUE)` reports the stored budget, OpenMP runtime settings and available BLAS information. Environment-variable values alone do not verify the current runtime settings.

Value

An invisible list containing the clamped integer setting returned by the compiled routine. The function is called for its effect on the thread configuration.

See Also

`st.get.num.threads`, `gs.set.num.threads`

`st.set.openblas.threads`

Sets the linked BLAS thread count

Description

Sets the linked BLAS thread count.

Usage

```
st.set.openblas.threads(active)
```

Arguments

active	One finite whole number between 1 and 256.

Details

The package and OpenMP thread budgets remain unchanged. The linked BLAS backend must expose supported thread controls and permit an independent setting. An unavailable control or incompatible shared OpenMP setting produces an error.

Value

An invisible `list` returned by the compiled 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 compiled 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. Further details appear in 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` *Sets 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 compiled 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 *Sets 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 compiled 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.

Imported individual identifiers and full stored pedigrees are copied into owned simulation strings. If no pedigree is attached, the imported name is the founder pedigree. Removing or changing the source marker dataset does not change those strings. No fitted symbolic-A prediction state is transferred.

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 compiled 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 — *Initializes 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. Further details appear in 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. Further details appear in 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 *Starts the SelectionTools timing helper*

Description

Starts the SelectionTools timing helper.

Usage

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

Arguments

depth	Timer depth, a finite whole number from 1 through 5.

Value

An invisible list returned by the compiled 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 *Stops the SelectionTools timing helper*

Description

Stops the SelectionTools timing helper.

Usage

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

Arguments

depth	Timer depth, a finite whole number from 1 through 5.
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` *Converts 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 optional `allele.codes` attribute is a list of integer coding vectors, named and ordered by marker ID. It records the internal allele codes corresponding to REF and successive ALT entries. STvcf exchange retains this attribute so fixed or subsetted loci keep the same coding as their fitted marker reference. The loader validates it before replacing a destination. Conversions preserve it; a manually edited marker list must keep it aligned. It is not part of plain-text VCF exchange.

The optional versioned `pedigree` attribute is a data frame with `name` and `pedigree` columns and integer attribute `version=1L`. Its names must match the individual identifiers in order. Without it, individuals are founders. Full strings are owned independently of ancestor populations. The optional `prediction.state` attribute preserves a compatible active symbolic-A fitted state during STvcf/RDS exchange. Conversion in either direction retains both attributes; loading validates them before replacing the destination.

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 identifiers must be unique, non-empty, and shorter than 256 bytes. Individual identifiers must be unique and non-empty; they use dynamically allocated strings subject to the R string-size limit. Whitespace and control characters are rejected; ordinary punctuation such as dots, underscores, and hyphens is permitted. 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 *Calculates 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; complete=TRUE prints 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 — *Writes linkage-map information for a SelectionTools data set*

Description

Writes 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, map information is also returned as a data frame assembled in memory. 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 selected 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.

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

Further details appear in 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` *Enables or controls SelectionTools console output*

Description

Enables or controls 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-dentmaize-vcf` *STvcf Marker Data for Dent Maize Parents*

Description

Lossless STvcf marker data for 123 parents and 35478 markers in the Melchinger and Frisch (2023) hybrid-prediction example data set.

Usage

```
data("v-dentmaize-vcf", package = "SelectionTools")
```

Format

A nine-component list of class `"STvcf"`:

`CHROM` 35478 missing integer chromosome codes.

`POS` 35478 missing double-precision map positions.

`ID` Unique marker identifiers in original marker order.

`REF`, `ALT` Allele labels `"1"` and `"2"`.

`individual` 123 parental identifiers, matching the hybrid records.

`allele1`, `allele2` 35478 by 123 raw matrices. Raw 0 denotes the reference allele and raw 1 the alternative allele. The two matrices agree because the supplied parents are homozygous.

`CHROM.levels` Empty character vector, explicitly indicating marker-only data without a linkage map.

Details

`st.load.vcf.data` loads `v.dentmaize.vcf` as a SelectionTools marker data set. The supplied object preserves the original marker genotypes and identifiers. No linkage map is supplied and no artificial positions are assigned. Relationship matrices depend on the marker genotypes and do not require a map; map-dependent analyses require separately supplied genuine map information.

The dent pool contains 29108 polymorphic biallelic markers. The stored `v.dentmaize.vcf` object retains all 35478 markers.

The `dataset.name`, `model.reference` and `source` attributes distinguish the Melchinger and Frisch (2023) example label from the original marker-data provenance.

Source

Model reference: Melchinger AE, Frisch M (2023). Genomic prediction in hybrid breeding: II. Reciprocal recurrent genomic selection with full-sib and half-sib families. Theoretical and Applied Genetics 136:203. doi:10.1007/s00122023044463. Original parental markers and hybrid records: Technow et al. (2014), Genetics 197:1343–1355. doi:10.1534/genetics.114.165860.

See Also

`v-flintmaize-vcf`, `v-hybmaize-phe`, `st.load.vcf.data`, `st.restrict.marker.data`, `st.get.A`

---

v-dentmaize2-vcf *Mapped Common-Marker Data for Dent Maize Parents*

Description

STvcf marker data for 123 dent maize parents at 8072 mapped markers that are polymorphic in both parental pools of the Melchinger and Frisch (2023) hybrid-prediction example.

Usage

```
data("v-dentmaize2-vcf", package = "SelectionTools")
```

Format

A nine-component list of class `"STvcf"`:

`CHROM` 8072 integer chromosome codes from 1 to 10.

`POS` 8072 double-precision estimated genetic positions in cM.

`ID` 8072 unique marker identifiers, retained from `v.dentmaize.vcf`.

`REF`, `ALT` Allele labels `"1"` and `"2"`.

`individual` 123 parental identifiers, unchanged from `v.dentmaize.vcf` and matching `v.hybmaize2.phe`.

`allele1`, `allele2` 8072 by 123 raw matrices. Raw 0 represents REF and raw 1 represents ALT. All calls are complete and homozygous, so the matrices are identical.

`CHROM.levels` The chromosome labels `"1"`, ..., `"10"`.

Details

The parental genotypes, allele coding and individual order are retained from `v.dentmaize.vcf`. Markers are selected from the original 35478-marker panel by requiring both a map position and polymorphism among the 86 flint parents and among the 123 dent parents. Both mapped parental objects contain the same 8072 markers in the same chromosome/position order. No additional marker filtering is needed to remove monomorphic loci.

The map matches `name_50k` to the existing marker identifiers with `make.names`, uses `chr` as the chromosome, and retains `pos_cM_estimated` as double-precision cM positions. The 10726 mapped markers in the original panel contain 8072 that are polymorphic in both pools. The supplied archive contributes the map; parental genotypes and identifiers come from the existing SelectionTools data.

Two exact within-chromosome position ties are retained in the stored objects. `st.load.vcf.data` separates ties internally by small deterministic offsets and leaves the stored STvcf objects unchanged. The `map.source` and `marker.selection` attributes describe the map provenance and the marker-selection rule. The genotype source and model reference are retained separately in the original attributes.

Source

Parental genotypes: Technow et al. (2014), Genetics 197:1343–1355. doi:10.1534/genetics.114.165860. Model reference: Melchinger AE, Frisch M (2023). Genomic prediction in hybrid breeding: II. Reciprocal recurrent genomic selection with full-sib and half-sib families. Theoretical and Applied Genetics 136:203. doi:10.1007/s00122023044463. Map: supplied 'maize2.tar.gz', 'map_HOH_pruned.csv', columns `name_50k`, `chr`, and `pos_cM_estimated`; these are exported unchanged in 'maize2.mmp'.

See Also

`v-flintmaize2-vcf`, `v-hybmaize2-phe`, `v-dentmaize-vcf`, `st.load.vcf.data`, `st.set.simpop`

`v-flintmaize-vcf` *STvcf Marker Data for Flint Maize Parents*

Description

Lossless STvcf marker data for 86 parents and 35478 markers in the Melchinger and Frisch (2023) hybrid-prediction example data set.

Usage

```
data("v-flintmaize-vcf", package = "SelectionTools")
```

Format

A nine-component list of class `"STvcf"`:

`CHROM` 35478 missing integer chromosome codes.

`POS` 35478 missing double-precision map positions.

`ID` Unique marker identifiers in original marker order.

`REF`, `ALT` Allele labels `"1"` and `"2"`.

`individual` 86 parental identifiers, matching the hybrid records.

`allele1`, `allele2` 35478 by 86 raw matrices. Raw 0 denotes the reference allele and raw 1 the alternative allele. The two matrices agree because the supplied parents are homozygous.

`CHROM.levels` Empty character vector, explicitly indicating marker-only data without a linkage map.

Details

`st.load.vcf.data` loads `v.flintmaize.vcf` as a SelectionTools marker data set. The supplied object preserves the original marker genotypes and identifiers. No linkage map is supplied and no artificial positions are assigned. Relationship matrices depend on the marker genotypes and do not require a map; map-dependent analyses require separately supplied genuine map information.

The flint pool contains 28240 polymorphic biallelic markers. The stored `v.flintmaize.vcf` object retains all 35478 markers.

The `dataset.name`, `model.reference` and `source` attributes distinguish the Melchinger and Frisch (2023) example label from the original marker-data provenance.

Source

Model reference: Melchinger AE, Frisch M (2023). Genomic prediction in hybrid breeding: II. Reciprocal recurrent genomic selection with full-sib and half-sib families. Theoretical and Applied Genetics 136:203. doi:10.1007/s00122023044463. Original parental markers and hybrid records: Technow et al. (2014), Genetics 197:1343–1355. doi:10.1534/genetics.114.165860.

See Also

`v-dentmaize-vcf`, `v-hybmaize-phe`, `st.load.vcf.data`, `st.restrict.marker.data`, `st.get.A`

---

v-flintmaize2-vcf *Mapped Common-Marker Data for Flint Maize Parents*

Description

STvcf marker data for 86 flint maize parents at 8072 mapped markers that are polymorphic in both parental pools of the Melchinger and Frisch (2023) hybrid-prediction example.

Usage

```
data("v-flintmaize2-vcf", package = "SelectionTools")
```

Format

A nine-component list of class "STvcf":

CHROM 8072 integer chromosome codes from 1 to 10.

POS 8072 double-precision estimated genetic positions in cM.

ID 8072 unique marker identifiers, retained from v.flintmaize.vcf.

REF, ALT Allele labels "1" and "2".

individual 86 parental identifiers, unchanged from v.flintmaize.vcf and matching v.hybmaize2.phe.

allele1, allele2 8072 by 86 raw matrices. Raw 0 represents REF and raw 1 represents ALT. All calls are complete and homozygous, so the matrices are identical.

CHROM.levels The chromosome labels "1", ..., "10".

Details

The parental genotypes, allele coding and individual order are retained from v.flintmaize.vcf. Markers are selected from the original 35478-marker panel by requiring both a map position and polymorphism among the 86 flint parents and among the 123 dent parents. Both mapped parental objects contain the same 8072 markers in the same chromosome/position order. No additional marker filtering is needed to remove monomorphic loci.

The map matches name_50k to the existing marker identifiers with make.names, uses chr as the chromosome, and retains pos_cM_estimated as double-precision cM positions. The 10726 mapped markers in the original panel contain 8072 that are polymorphic in both pools. The supplied archive contributes the map; parental genotypes and identifiers come from the existing SelectionTools data.

Two exact within-chromosome position ties are retained in the stored objects. st.load.vcf.data separates ties internally by small deterministic offsets and leaves the stored STvcf objects unchanged. The map.source and marker.selection attributes describe the map provenance and the marker-selection rule. The genotype source and model reference are retained separately in the original attributes.

Source

Parental genotypes: Technow et al. (2014), Genetics 197:1343–1355. doi:10.1534/genetics.114.165860. Model reference: Melchinger AE, Frisch M (2023). Genomic prediction in hybrid breeding: II. Reciprocal recurrent genomic selection with full-sib and half-sib families. Theoretical and Applied Genetics 136:203. doi:10.1007/s00122023044463. Map: supplied 'maize2.tar.gz', 'map_HOH_pruned.csv', columns name_50k, chr, and pos_cM_estimated; these are exported unchanged in 'maize2.mmp'.

See Also

v-dentmaize2-vcf, v-hybmaize2-phe, v-flintmaize-vcf, st.load.vcf.data, st.set.simpop

`v-hybmaize-phe` *Phenotype Data for Maize Hybrids*

Description

Phenotype records for 1254 crosses between dent and flint maize parents in the Melchinger and Frisch (2023) hybrid-prediction example data set. Parental identifiers match the accompanying marker data.

Usage

```
data("v-hybmaize-phe")
```

Format

A data frame with 1254 rows and six columns:

`hybrid` Hybrid identifier, with parents separated by a period.

`dent` Dent parent identifier.

`flint` Flint parent identifier.

`GY` Grain yield phenotype, as supplied in the source data.

`GM` Grain moisture phenotype, as supplied in the source data.

`hy` Hybrid identifier, with parents separated by a colon.

Details

The `dataset.name`, `model.reference` and `source` attributes retain the example label, model reference and original data provenance separately.

Source

Model reference: Melchinger AE, Frisch M (2023). Genomic prediction in hybrid breeding: II. Reciprocal recurrent genomic selection with full-sib and half-sib families. Theoretical and Applied Genetics 136:203. doi:10.1007/s00122023044463. Original parental markers and hybrid records: Technow et al. (2014), Genetics 197:1343–1355. doi:10.1534/genetics.114.165860.

v-hybmaize2-phe *Hybrid Phenotypes for the Mapped Maize Parental Data*

Description

Phenotype records for 1254 crosses between the 86 flint and 123 dent parents in `v.flintmaize2.vcf` and `v.dentmaize2.vcf`.

Usage

```
data("v-hybmaize2-phe", package = "SelectionTools")
```

Format

A data frame with 1254 rows and six columns:

`hybrid` Hybrid identifier, with parents separated by a period.

`dent` Dent parent identifier matching `v.dentmaize2.vcf`.

`flint` Flint parent identifier matching `v.flintmaize2.vcf`.

`GY` Grain yield phenotype, as supplied in the source data.

`GM` Grain moisture phenotype, as supplied in the source data.

`hy` Hybrid identifier, with parents separated by a colon.

Details

This object is an unchanged copy of `v.hybmaize.phe`, including all values, identifiers, row order, column types and attributes. Every parent has marker data in the corresponding mapped parental object. The marker subset changes the genomic information available for fitting, so a model using the mapped panel is fitted to that panel rather than borrowing estimates from the larger marker panel.

Direct storage of these records requires marker-data names `"flint"` and `"dent"`, matching the parental column names in the stored performance frame.

Source

Melchinger AE, Frisch M (2023). Genomic prediction in hybrid breeding: II. Reciprocal recurrent genomic selection with full-sib and half-sib families. Theoretical and Applied Genetics 136:203. doi:10.1007/s00122023044463. Original parental markers and hybrid records: Technow et al. (2014), Genetics 197:1343–1355. doi:10.1534/genetics.114.165860.

See Also

`v-flintmaize2-vcf`, `v-dentmaize2-vcf`, `v-hybmaize-phe`, `st.hybrid.data.set`, `st.lmm`

v-lmm-climate *Balanced Climate and Variety Experiment*

Description

A balanced experiment with two varieties, three climates and four observations in each variety by climate combination.

Usage

```
data("v-lmm-climate", package = "SelectionTools")
```

Format

A `data.frame` named `climate` with 24 rows and 5 columns.

`variety` Variety identifier. Factor with levels a and b.

`climate` Climate identifier. Factor with levels I, II and III.

`cv` Variety by climate combination. Factor with six levels.

`rep` Replicate identifier. Factor with levels r1 to r4.

`yield` Yield response. Integer.

Details

The stored data contain no missing values.

Value

Loading the dataset creates the `climate` data frame.

See Also

`st.lmm`, `st.lsm`, `v-lmm-data`

v-lmm-corn *Corn Yield for a One-Sample Analysis*

Description

Corn yield observations used to illustrate the sample mean, sample variance, standard error and confidence interval for a mean.

Usage

```
data("v-lmm-corn", package = "SelectionTools")
```

Format

A `data.frame` named `corn` with 200 rows and 1 column.

`yld` Corn yield in bushels per acre. Integer.

Details

The stored data contain no missing values.

Value

Loading the dataset creates the `corn` data frame.

See Also

`st.lmm`, `st.lsm`, `v-lmm-data`

---

v-lmm-data *Datasets for Linear Mixed Models*

Description

Agricultural and repeated-measures datasets for linear mixed models. Dataset entries describe the loaded objects, variables, storage types, factor levels and missing values. The additional datasets for moment estimators are documented separately in `v-lmm-climate`, `v-lmm-corn`, `v-lmm-iris-two`, `v-lmm-iris`, `v-lmm-fertilizer-regression`, `v-lmm-fertilizer-replicated`, `v-lmm-rats` and `v-lmm-tomato`. The machine and operator experiment is documented in `v-lmm-machines`. The additional regression dataset `v-lmm-mtcars` is documented in its own help topic and loads the numeric data frame `mtcars`.

Usage

```
data("v-lmm-oats", package = "SelectionTools")
```

Format

Each file contains one `data.frame`. Row counts, storage types and factor levels describe the stored data. Column names are case-sensitive.

`v-lmm-soils` Yield observations from fields within villages and regions. File '`v-lmm-soils.rda`' loads `soils`, with 24 rows and 4 columns. The stored data contain no missing values.

`region` Region identifier. Factor with 3 levels: `I`, `II`, `III`.

`village` Village identifier. Factor with 6 levels: `A` to `F`. Villages `A` and `B` belong to region `I`, `C` and `D` to region `II`, and `E` and `F` to region `III`.

`field` Field identifier. Factor with 4 levels: `f1`, `f2`, `f3`, `f4`.

`yield` Yield response. Integer.

`v-lmm-rapeseed` Rapeseed height observations across environments. File '`v-lmm-rapeseed.rda`' loads `rapeseed`, with 120 rows and 4 columns. The stored data contain no missing values.

`geno` Genotype identifier. Factor with 20 levels.

`env` Environment identifier. Factor with 2 levels: `e1`, `e2`.

`rep` Replicate identifier. Factor with 3 levels: `r1`, `r2`, `r3`.

`hgt` Height response. Integer.

`v-lmm-soybean-blocks` Soybean yield observations in an incomplete-block layout. File '`v-lmm-soybean-blocks.rda`' loads `soybean.blocks`, with 50 rows and 5 columns. The stored data contain no missing values.

`Rep` Replicate identifier. Factor with 2 levels: `r1`, `r2`.

`Block` Block identifier. Factor with 10 levels.

`Plot` Plot identifier. Factor with 5 levels: `p1`, `p2`, `p3`, `p4`, `p5`.

`Variety` Variety identifier. Factor with 25 levels.

`Yield` Yield response. Integer.

`v-lmm-alfalfa` Balanced alfalfa split-plot experiment: varieties are whole-plot treatments within blocks and dates are subplot treatments. File '`v-lmm-alfalfa.rda`' loads `alfalfa`, with 72 rows and 4 columns. The stored data contain no missing values.

`Variety` Variety identifier. Factor with 3 levels: `Cossack`, `Ladack`, `Ranger`.

`Date` Date treatment. Factor with 4 levels: `A`, `B`, `C`, `D`.

`Block` Block identifier. Factor with 6 levels: `b1`, `b2`, `b3`, `b4`, `b5`, `b6`.

`Yield` Yield response. Numeric (double).

`v-lmm-oats` Balanced oats split-plot experiment: varieties are whole-plot treatments within blocks and nitrogen levels are subplot treatments. File '`v-lmm-oats.rda`' loads `oats`, with 72 rows and 4 columns. The stored data contain no missing values.

`Block` Block identifier. Factor with 6 levels: `I`, `II`, `III`, `IV`, `V`, `VI`.

`Variety` Variety identifier. Factor with 3 levels: `Golden Rain`, `Marvellous`, `Victory`.

`Nitro` Nitrogen treatment. Factor with 4 levels: `n0.0`, `n0.2`, `n0.4`, `n0.6`.

`Yield` Yield response. Integer.

`v-lmm-barley-lattice` Barley yield observations with row and column blocks. File '`v-lmm-barley-lattice.rda`' loads `barley.lattice`, with 150 rows and 7 columns. These observations are from the 1976 balanced lattice square trial at Slate Hall Farm, UK, described by Gilmour, Thompson and Cullis (1995). The six design and treatment columns are stored as factors; the zero-padded field coordinates require no releveling before spatial analysis. The stored data contain no missing values.

`Rep` Replicate identifier. Factor with 6 levels: `re1`, `re2`, `re3`, `re4`, `re5`, `re6`.

`RowBlk` Row-block identifier. Factor with 30 levels.

`ColBlk` Column-block identifier. Factor with 30 levels.

`Row` Physical field-row identifier. Factor with levels `row01`, ..., `row10`, in physical and lexical order.

`Column` Physical field-column identifier. Factor with levels `col01`, ..., `col15`, in physical and lexical order.

`Variety` Variety identifier. Factor with 25 levels.

`yield` Yield response. Integer.

`v-lmm-rapeseed-blocks` Rapeseed height observations with alternative block layouts. File 'v-lmm-rapeseed-blocks.rda' loads `rapeseed.blocks`, with 120 rows and 7 columns. The stored data contain no missing values.

`rep` Replicate identifier. Factor with 3 levels: `r1`, `r2`, `r3`.

`blk` Block identifier. Factor with 10 levels.

`geno` Genotype identifier. Factor with 20 levels.

`RowBlk` Row-block identifier. Factor with 6 levels: `rb01`, `rb02`, `rb03`, `rb04`, `rb05`, `rb06`.

`ColBlk` Column-block identifier. Factor with 20 levels.

`env` Environment identifier. Factor with 2 levels: `k`, `s`.

`hgt` Height response. Integer.

`v-lmm-maize-counties` Maize yield observations from trials in several counties. File 'v-lmm-maize-counties.rda' loads `maize.counties`, with 1152 rows and 7 columns. The stored data contain no missing values.

`county` County identifier. Factor with 6 levels: `C1`, `C2`, `C3`, `C4`, `C5`, `C6`.

`row` Row identifier. Factor with 18 levels.

`col` Column identifier. Factor with 11 levels.

`rep` Replicate identifier. Factor with 3 levels: `R1`, `R2`, `R3`.

`block` Block identifier. Factor with 8 levels: `B1`, `B2`, `B3`, `B4`, `B5`, `B6`, `B7`, `B8`.

`yield` Yield response. Numeric (double).

`gen` Genotype identifier. Factor with 64 levels.

`v-lmm-soybean` Soybean yield observations across locations and years. File 'v-lmm-soybean.rda' loads `soybean`, with 306 rows and 6 columns. The stored data contain no missing values.

`yield` Yield response. Numeric (double).

`rep` Replicate identifier. Factor with 4 levels: `R1`, `R2`, `R3`, `R4`.

`gen` Genotype identifier. Factor with 7 levels: `Chip`, `Cors`, `Evan`, `Hodg`, `S200`, `Well`, `Wilk`.

`env` Environment identifier. Factor with 12 levels.

`loc` Location identifier. Factor with 4 levels: `A`, `G`, `L`, `V`.

`year` Year. Factor with 3 levels: `y80`, `y81`, `y82`.

`v-lmm-soybean-met` An extended soybean multi-environment yield dataset. File 'v-lmm-soybean-met.rda' loads `soybean.met`, with 1454 rows and 6 columns. The stored data contain no missing values.

`yield` Yield response. Numeric (double).

`rep` Replicate identifier. Factor with 4 levels: `R1`, `R2`, `R3`, `R4`.

`gen` Genotype identifier. Factor with 7 levels: `Chip`, `Cors`, `Evan`, `Hodg`, `S200`, `Well`, `Wilk`.

`env` Environment identifier. Factor with 55 levels.

`loc` Location identifier. Factor with 10 levels.

`year` Year. Factor with 12 levels.

`v-lmm-material-groups` Multi-environment observations with local and reference material groups. File 'v-lmm-material-groups.rda' loads `material.groups`, with 464 rows and 11 columns. The stored data contain no missing values.

`env` Environment identifier. Factor with 8 levels: `B70`, `B71`, `L70`, `L71`, `N70`, `N71`, `R70`, `R71`.

`loc` Location identifier. Factor with 4 levels: `Brookstead`, `Lawes`, `Nambour`, `RedlandBay`.

year Year. Factor with 2 levels: 1970, 1971.

gen Genotype identifier. Factor with 58 levels.

yield Yield response. Numeric (double).

height Height response. Numeric (double).

lodging Lodging response. Numeric (double).

size Recorded size trait. Numeric (double).

protein Protein response. Numeric (double).

oil Oil response. Numeric (double).

grp Material group (local or reference). Factor with 2 levels: local, ref.

v-lmm-maize-met Maize plot records for yield, dry-matter content and plant height across environments. The analyses estimate adjusted genotype means and genetic covariances among traits. File 'v-lmm-maize-met.rda' loads maize.met, with 1200 rows and 6 columns. The stored data contain no missing values.

env Environment identifier. Factor with 24 levels.

gen Genotype identifier. Factor with 25 levels.

rep Replicate identifier. Factor with 2 levels: R1, R2.

pht Plant-height response. Integer.

dmc Dry-matter-content response. Numeric (double).

yld Yield response. Numeric (double).

v-lmm-barley-met Anonymized barley multi-environment observations. File 'v-lmm-barley-met.rda' loads barley.met, with 900 rows and 9 columns: 15 lines in 5 locations, 3 years and 4 replicates. The stored data contain no missing values. The record for HFH, y2018, lne04, r3 contains trait means from the other three replicates in that cell to complete the balanced layout. Location and line identifiers are coded.

loc Location identifier. Factor with levels HOH, EWE, EIN, OLI, HFH, in that order.

year Year. Factor with levels y2016, y2017, y2018.

gen Line identifier. Factor with levels lne01 to lne15.

rep Replicate identifier. Factor with levels r1 to r4.

DHE Recorded DHE trait. Numeric.

DMA Recorded DMA trait. Numeric.

PLH Plant-height response. Numeric.

GYLD Grain-yield response. Numeric.

TKW Thousand-kernel-weight response. Numeric.

v-lmm-met-unbalanced Yield observations from unbalanced multi-environment trials. File 'v-lmm-met-unbalanced.rda' loads met.unbalanced, with 450 rows and 6 columns. The stored data contain no missing values.

gen Genotype identifier. Factor with 15 levels and zero-padded identifiers such as g01.

rep Replicate identifier. Factor with 3 levels: r1, r2, r3.

env Environment identifier. Factor with 10 levels.

yld Yield response. Numeric (double).

loc Location identifier. Factor with 6 levels: ASE, BOV, MOS, NIE, RHH, ROS.

year Year. Factor with 2 levels: 2015, 2016.

`v-lmm-met-missing` Multi-trait observations with missing yield values. File '`v-lmm-met-missing.rda`' loads `met.missing`, with 720 rows and 8 columns.

- `loc` Location identifier. Factor with 5 levels: `loc1`, `loc2`, `loc3`, `loc4`, `loc5`.
- `rep` Replicate identifier. Factor with 4 levels: `r1`, `r2`, `r3`, `r4`.
- `year` Year. Factor with 3 levels: `y2017`, `y2018`, `y2019`.
- `gen` Genotype identifier. Factor with 20 levels.
- `DHE` Recorded DHE trait. Integer.
- `DMA` Recorded DMA trait. Integer.
- `PLH` Plant-height response. Numeric (double).
- `YLD` Yield response. Numeric (double). Contains 2 missing values (`NA`).

`v-lmm-met-seven-traits` Seven recorded traits across locations and years. File '`v-lmm-met-seven-traits.rda`' loads `met.seven.traits`, with 300 rows and 11 columns.

- `year` Year. Factor with 2 levels: `y2019`, `y2020`.
- `loc` Location identifier. Factor with 3 levels: `loc1`, `loc2`, `loc3`.
- `rep` Replicate identifier. Factor with 3 levels: `r1`, `r2`, `r3`.
- `gen` Genotype identifier. Factor with 28 levels.
- `YLD` Yield response. Numeric (double).
- `HLW` Hectolitre-weight response. Numeric (double). Contains 9 missing values (`NA`).
- `TKW` Thousand-kernel-weight response. Numeric (double).
- `PLH` Plant-height response. Numeric (double).
- `DH` Recorded DH trait. Integer.
- `DM` Recorded DM trait. Integer.
- `SC` Recorded SC score. Integer.

`v-lmm-barley-seven-traits` Barley data with all seven responses observed. File '`v-lmm-barley-seven-traits.rda`' loads `barley.seven.traits`, with 291 rows and 14 columns. It contains the records from `met.seven.traits` with all seven responses observed. The nine records with missing `HLW` are excluded. Three factors describe the groups used in the fit. `env` identifies the five year-location combinations, `gen_env` identifies genotype within environment, and `env_rep` identifies replicate within environment. The original measurement units and row identifiers are retained.

`v-lmm-chickens` Repeated chick body-weight measurements under different diets. File '`v-lmm-chickens.rda`' loads `chickens`, with 578 rows and 4 columns. The stored data contain no missing values.

- `weight` Body-weight response. Integer.
- `Time` Measurement time. Integer.
- `Chick` Chick identifier. Factor with 50 levels.
- `Diet` Diet treatment. Factor with 4 levels: `d1`, `d2`, `d3`, `d4`.

Details

The file name and loaded object name are listed for every dataset. All listed datasets are available through `data()`; its `envir` argument specifies the destination environment. Object names follow the names listed above, with dots separating words in multiword names.

All categorical grouping variables, including years, are stored as factors. Responses and continuous covariates (including chick measurement time) remain numeric. Observations and missing values

are retained except for constructions explicitly described above. The barley multi-environment identifiers are coded, and its balancing record is documented in its dataset entry. Measurement units are retained; no unit conversion or standardization is applied when storing these datasets. Trait abbreviations without a recorded definition are used as variable names.

The stored missing values occur in `YLD` of `met.missing` (two values) and `HLW` of `met.seven.traits` (nine values). Missing experimental combinations in an unbalanced design are distinct from explicit `NA` values. Repeated group labels can identify different groups across environments or higher-level factors. The model formula must specify the required interactions and nesting.

Value

Each file supplies the named data frame described above. `data` loads the data frame into the environment specified by `envir`.

Constructed tropical maize trial

The additional file '`v-lmm-tropmaize.rda`' loads `tropmaize`, a 792-row data frame with columns `block`, `plot`, `gen` and `y`. It contains synthetic observations from three complete blocks in one environment, with all 264 genotypes in every block. Field-only genotype BLUEs reproduce the values in `v-tropmaize-phe`. The construction and full format are described in `v-lmm-tropmaize`. The data contain no missing values.

Additional experimental datasets

`v-lmm-fertilizer` Object `fertilizer`: 25 observations of `yield`, fertilizer factor `fert` and replication factor `rep`.

`v-lmm-cotton` Object `cotton`: 15 observations of `Strength`, block factor `Blk` and fertilizer factor `Fert`.

`v-lmm-sugar` Object `sugar`: 36 observations of `Yield` and factors `Row`, `Col` and `Variety`.

`v-lmm-peas` Object `peas`: 36 observations of `Yield` and factors `Variety`, `Spacing` and `Block`.

References

Gilmour AR, Thompson R, Cullis BR (1995). Average information REML: An efficient algorithm for variance parameter estimation in linear mixed models. Biometrics 51:1440–1450. doi:10.2307/2533274.

See Also

`st.lmm`, `st.lsm`

v-lmm-fertilizer-regression

Fertilizer Dose and Yield for Linear Regression

Description

Nine pairs of fertilizer-dose and crop-yield observations for simple linear regression.

Usage

```
data("v-lmm-fertilizer-regression", package = "SelectionTools")
```

Format

A `data.frame` named `fertilizer.regression` with 9 rows and 2 columns.

`dueng` Fertilizer dose. Numeric (double).

`ertrag` Crop yield. Numeric (double).

Details

Measurement units are not recorded. The stored data contain no missing values.

Value

Loading the dataset creates the `fertilizer.regression` data frame.

See Also

`st.lmm`, `st.lsm`, `v-lmm-data`

v-lmm-fertilizer-replicated

Replicated Fertilizer Doses for a Regression Lack-of-Fit Test

Description

Eleven yield observations at four fertilizer doses, used to partition regression residual variation into lack of fit and pure error.

Usage

```
data("v-lmm-fertilizer-replicated", package = "SelectionTools")
```

Format

A `data.frame` named `fertilizer.replicated` with 11 rows and 2 columns.

`dueng` Numeric fertilizer dose, with values 1, 3, 5 and 7. The first three doses have three observations each; dose 7 has two.

`ertrag` Numeric crop yield.

Details

Measurement units are not recorded. There are no missing values. Replication at the four dose levels provides 7 degrees of freedom for pure error. A linear mean function has 2 degrees of freedom for lack of fit relative to unrestricted dose means.

Value

Loading the dataset creates the `fertilizer.replicated` data frame.

---

v-lmm-halfdiallel *Maize Half Diallel for Shared Parental GCA and Cross SCA*

Description

Sugar measurements for 21 unordered crosses among seven maize parents, with two technical repetitions per cross. These are the small half-diallel data used in the quantitative-genetics vignette of **lme4breeding**.

Usage

```
data("v-lmm-halfdiallel", package = "SelectionTools")
```

Format

Loads `halfdiallel`, a data frame with 42 rows and five variables:

`rep` Integer technical-repetition identifier, 1 or 2.

`male` Male-parent factor with all seven parent identifiers as levels.

`female` Female-parent factor with the same seven levels as male.

`cross` Factor identifying the 21 unordered parental crosses, labelled as `P1:P2`, for example.

`sugar` Numeric sugar-content response in the source units.

There are no missing values. All 21 unordered pairs of distinct parents occur twice; self crosses and reciprocal observations are absent.

Details

Response values and repetition labels are unchanged from `DT_halfdiallel`, distributed with **enhancer** version 1.1.2 and used by **lme4breeding**. The object is renamed for the SelectionTools vignette, with a source attribute added. Both parental roles are factors with common levels P1 through P7. The unordered cross factor is included, so the model needs no preprocessing. The redundant source column `geno` is omitted: `cross` identifies the same crosses and supplies the SCA term. It is self-contained: neither source package is needed to load it.

Value

A data frame named `halfdiallel`, loaded into the selected environment by `data`.

Source

Giovanny Covarrubias-Pazaran, **enhancer** 1.1.2, `DT_halfdiallel`; quantitative-genetics vignette of **lme4breeding**. The source package declares GPL version 2 or later.

References

Covarrubias-Pazaran G (2016) Genome-assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11:e0156744. doi:10.1371/journal.pone.0156744

`v-lmm-iris` *Iris Measurements for MANOVA*

Description

Four measurements on 150 Iris plants from three species, used for multivariate analysis of variance.

Usage

```
data("v-lmm-iris", package = "SelectionTools")
```

Format

A `data.frame` named `iris.manova` with 150 rows and 5 columns.

`SepalLength` Sepal length in centimetres. Numeric (double).

`SepalWidth` Sepal width in centimetres. Numeric (double).

`PetalLength` Petal length in centimetres. Numeric (double).

`PetalWidth` Petal width in centimetres. Numeric (double).

`Species` Species identifier. Factor with levels setosa, versicolor and virginica.

Details

The measurements agree with the standard Iris dataset. The stored data contain no missing values.

Value

Loading the dataset creates the `iris.manova` data frame.

See Also

`st.lmm`, `st.lsm`, `v-lmm-data`

`v-lmm-iris-two` *Two Iris Species for Hotelling's Test*

Description

Sepal measurements on ten plants from each of two Iris species, used for the pooled-covariance Hotelling two-sample test.

Usage

```
data("v-lmm-iris-two", package = "SelectionTools")
```

Format

A `data.frame` named `iris.two` with 20 rows and 3 columns.

`SepalLength` Sepal length in centimetres. Numeric (double).

`SepalWidth` Sepal width in centimetres. Numeric (double).

`Species` Species identifier. Factor with levels versicolor and virginica.

Details

The stored data contain no missing values.

Value

Loading the dataset creates the `iris.two` data frame.

See Also

`st.lmm`, `st.lsm`, `v-lmm-data`

v-lmm-machines *Machine and Operator Experiment*

Description

Performance scores from two machine types evaluated twice by each of three operators. The crossed design illustrates inference spaces for least-squares means in linear mixed models.

Usage

```
data("v-lmm-machines", package = "SelectionTools")
```

Format

A `data.frame` named `machines` with 12 rows and 3 columns.

`machine` Machine type. Factor with levels `1` and `2`.

`operator` Operator identifier. Factor with levels `1`, `2` and `3`.

`score` Observed performance score. Numeric (double).

Details

Each machine and operator combination has two observations. The stored data contain no missing values.

Value

Loading the dataset creates the `machines` data frame.

Source

Sanders, W. L. (1991). The use of the SAS system in statistical training. Proceedings of the Sixteenth Annual SAS Users Group International Conference. https://support.sas.com/resources/papers/proceedings-archive/SUGI91/Sugi-91-283%20Sanders.pdf

References

McLean, R. A., Sanders, W. L. and Stroup, W. W. (1991). A unified approach to mixed linear models. The American Statistician, 45, 54–64. doi:10.1080/00031305.1991.10475767.

See Also

`st.lmm`, `st.lsm`, `v-lmm-data`

`v-lmm-mtcars` *Motor Trend Car Measurements for Regression*

Description

Fuel consumption and vehicle characteristics for 32 cars, used for ordinary regression and fixed-effect subset comparison.

Usage

```
data("v-lmm-mtcars", package = "SelectionTools")
```

Format

A `data.frame` named `mtcars` with 32 rows and 11 columns. Every column is stored as a numeric double. Car names are row names.

`mpg` Fuel economy in miles per US gallon.

`cyl` Number of engine cylinders.

`disp` Engine displacement in cubic inches.

`hp` Gross horsepower.

`drat` Rear axle ratio.

`wt` Vehicle weight in thousands of pounds.

`qsec` Quarter-mile time in seconds.

`vs` Engine configuration: 0 for V-shaped and 1 for straight.

`am` Transmission: 0 for automatic and 1 for manual.

`gear` Number of forward gears.

`carb` Number of carburettors.

Details

The observations concern 1973–1974 car models and were originally compiled from the US magazine *Motor Trend*. The data contain no missing values. Original values, row names, column names and row order are retained. The coded variables `cyl`, `vs`, `am`, `gear` and `carb` remain numeric; they are not automatically converted to factors.

The numeric values match the Dryad release *Car parameters*, dedicated to the public domain under CC0 1.0.

Value

Loading the dataset creates the `mtcars` data frame.

Source

Henderson H, Velleman PF (2016). Car parameters. Dryad dataset. DOI: 10.15146/R32S37. File 'CarParameters.tsv', released 28 September 2016.

Henderson HV, Velleman PF (1981). Building Multiple Regression Models Interactively. *Biometrics* 37:391–411.

---

v-lmm-rats *Rats for Partial and Sequential Regression Tests*

Description

Initial weight, feed intake and final weight for 35 rats, used to distinguish partial from sequential tests in multiple regression.

Usage

```
data("v-lmm-rats", package = "SelectionTools")
```

Format

A data.frame named rats with 35 rows and 3 columns.

x.1 Initial rat weight. Numeric (double).

x.2 Feed intake. Integer.

y Final rat weight. Numeric (double).

Details

Measurement units are not recorded. The stored data contain no missing values.

Value

Loading the dataset creates the rats data frame.

See Also

st.lmm, st.lsm, v-lmm-data

v-lmm-tomato *Tomato Plant Heights for Linear Hypotheses*

Description

Seven plant-height observations in three treatment groups, used for a cell-means model and explicit linear hypotheses.

Usage

```
data("v-lmm-tomato", package = "SelectionTools")
```

Format

A `data.frame` named `tomato` with 7 rows and 2 columns.

`y` Plant height. Numeric (double).

`x.v` Treatment identifier. Factor with levels 1, 2 and 3; group sizes are 3, 2 and 2.

Details

Plant-height measurement units are not recorded. The stored data contain no missing values.

Value

Loading the dataset creates the `tomato` data frame.

See Also

`st.lmm`, `st.lsm`, `v-lmm-data`

v-lmm-tropmaize *Constructed Tropical Maize Complete-Block Trial*

Description

Synthetic plot observations for the 264 tropical maize lines in `v-tropmaize-phe`, arranged in three complete blocks in one environment. The field-only mixed-model genotype BLUEs reproduce the original phenotypic values. The dataset illustrates field-trial and genomic analyses; it is not the original plot-level experiment.

Usage

```
data("v-lmm-tropmaize", package = "SelectionTools")
```

Format

A data frame named `tropmaize` with 792 rows and four columns:

`block` Complete-block factor with three levels, 1–3. Each block contains all 264 genotypes.

`plot` Integer plot position, 1–264, within each block.

`gen` Genotype factor with 264 levels. Labels match the individual identifiers in `v-tropmaize-phe` and `v-tropmaize-vcf`.

`y` Positive numeric plot response on the original phenotype scale. Values are not rounded. No additional measurement unit is assigned.

There are no missing observations. Each genotype occurs once in each block. All plots belong to one environment, so no environment column is needed. The `construction` attribute records the seed, design, nominal block effects, residual standard deviation and centring rule. The data file uses xz compression at level 9.

Details

The construction uses seed 20261014. Nominal block effects are -0.10, 0 and 0.10. Independent normal plot errors with nominal standard deviation 0.18 are generated and centred within each genotype across the three blocks. The centred errors and block effects are added to the original genotype values. For genotypes whose deviations would be too large, all three deviations are multiplied by the same factor to bound their absolute values by 0.85 times the target mean. This retains the genotype mean and keeps all responses positive. The bound and centring impose constraints on these synthetic observations; they are illustrative construction choices, not variance estimates or independent errors from the original study. Plots order is independently randomized within each complete block.

The genotype arithmetic means across the three complete blocks reproduce `v.tropmaize.phe$yy`, matched by `v.tropmaize.phe$ii`, to floating-point precision. With fixed genotype effects, a random block effect and a common residual variance, genotype BLUEs equal these arithmetic means, regardless of the positive block and residual variance components.

Genomic predictions are not constrained to reproduce these genotype means.

The data represent one environment; no `env` or `rep` column is stored.

Source

Constructed for the SelectionTools linear mixed-model vignette from the genotype identifiers and phenotypic values in `v-tropmaize-phe`. The original phenotype and marker source is Crossa et al. (2010), *Genetics* 186, 713–724, doi:10.1534/genetics.110.118521.

See Also

`st.lmm`, `st.lsm`, `gs.make.A`, `v-tropmaize-phe`, `v-tropmaize-vcf`

`v-lmm-tropmaize-lattice`
Tropical Maize Alpha-Lattice Trials in Five Environments

Description

A constructed tropical maize dataset with 264 genotypes evaluated in three replications at each of five environments. Each replication is an alpha-lattice with 22 incomplete blocks of 12 plots.

Usage

```
data("v-lmm-tropmaize-lattice", package = "SelectionTools")
```

Format

A data frame named `tropmaize.lattice` with 3960 rows and six columns:

`env` Environment factor with levels `E1` to `E5`.

`rep` Replication factor with three levels within each environment.

`block` Incomplete-block factor with 22 levels within each replication and environment.

`plot` Integer plot position, 1 to 12, within each incomplete block.

`gen` Genotype factor with 264 levels. Identifiers match those in `v-tropmaize-phe` and `v-tropmaize-vcf`.

`y` Positive numeric plot response on the scale of the supplied maize phenotypes. No additional measurement unit is assigned.

There are no missing observations. Every genotype occurs once in each replication, giving 15 observations per genotype and 792 per environment. The `construction` attribute retains the seed and construction settings.

Details

Replication and block labels are reused across environments. A replication is identified by `env:rep`, and an incomplete block by `env:rep:block`. Each pair of distinct genotypes occurs together in an incomplete block at most once within an environment. The treatment information matrix has rank 263 in every environment.

This is the original five-environment lattice dataset. It is available separately from `v-lmm-tropmaize`, the three-complete-block dataset in one environment. Loading this topic creates `tropmaize.lattice`.

The responses are synthetic, not observations from the original field experiment. The construction uses common genotype-to-block incidence across environments and centred contributions so that field-model genotype BLUEs recover the supplied phenotype values. These imposed constraints should be considered when interpreting analyses or using the data for benchmarks. The original observations and construction attributes are retained unchanged.

Source

Constructed for SelectionTools mixed-model examples from the genotype identifiers and phenotypic values in `v-tropmaize-phe`.

See Also

v-lmm-tropmaize, st.lmm, st.lsm, gs.make.A, v-tropmaize-phe, v-tropmaize-vcf

---

v-rapeseed *Rapeseed Marker and Flowering-Time Data for the GWAS Vignette*

Description

Rapeseed data for genome-wide association analysis, containing 22857 SNP markers for 193 rapeseed lines and flowering-time observations, provided by Rod Snowdon (University of Giessen). Genotypes use the compact STvcf representation; physical marker coordinates and phenotypes are separate objects.

Usage

```
data("v-rapeseed", package = "SelectionTools")
```

Format

The data file loads three objects:

`v.rapeseed.vcf` A nine-component list of class `"STvcf"` containing 22857 markers and 193 individuals. The `ID` and `individual` components retain the supplied unique marker and line identifiers in their original order. `REF` and `ALT` contain the neutral allele labels `"1"` and `"2"`. The `allele1` and `allele2` components are 22857 by 197 raw matrices using VCF allele indices 0 and 1. All calls are complete. `CHROM` is entirely `NA_integer_`, `POS` is entirely `NA_real_`, and `CHROM.levels` is `character()`, declaring marker-only data without a genetic linkage map. The `physical.map` attribute contains the physical map described below, with positions in base pairs and rows aligned exactly to `ID`. The `phased` attribute is `FALSE`.

`v.rapeseed.phe` A data frame with 193 rows and columns `gen` (genotype identifier) and `flt` (flowering time). Its row order matches `v.rapeseed.vcf$individual`. All 193 flowering-time values are observed. Lines `SW068`, `SW125`, `SW146`, and `SW195`, which lack phenotypes, are excluded from both marker and phenotype data. The measurement unit and time origin are not specified.

`v.rapeseed.map` A data frame with 22857 rows and columns `marker`, `chromosome`, and `position`. It preserves the source marker identifiers, numeric chromosome labels 1 through 19, and integer physical positions in base pairs. The `position.unit` attribute is `"bp"`. Rows match the marker order in the STvcf object.

Details

All 22857 markers and the 193 individuals with observed flowering time are retained in the packaged data. The four lines without flowering-time observations are excluded before packaging. No marker filtering, imputation, phasing, or phenotype replacement is performed.

The original genotype scores `-1`, `0`, and `1` are represented by unphased VCF calls `0/0`, `0/1`, and `1/1`, respectively. Numeric REF/ALT labels are used because nucleotide identities are not available.

With the default strictACTG = FALSE, st.load.vcf.data maps these allele indices to internal SelectionTools allele codes 1 and 2. The resulting default marker dosage is the original score plus one. No sign or allele frequency information is lost.

The source positions are physical coordinates, whereas STvcf map fields are reserved for genetic positions in centimorgans. Consequently the STvcf object contains no linkage map. Loading the STvcf object with st.load.vcf.data copies its physical.map attribute to compiled marker-data storage. Subsequent st.gwas calls use these coordinates automatically, without a separate map argument. The separately loaded v.rapeseed.map is identical to this attribute. The loader does not interpret the physical-map attribute as a genetic map. These data do not supply a genetic map for recombination or simulation.

Two within-chromosome coordinate ties, one on chromosome 5 and one on chromosome 6, are retained exactly. The tied markers have distinct identifiers. No positional offset, marker renaming or deduplication is applied to the physical map.

The three objects are stored together in 'v-rapeseed.rda' using xz compression at level 9. The STvcf source, original.coding, and marker.selection attributes record provenance and the lossless conversion rule. These data are distinct from the multi-environment height data loaded by v-lmm-rapeseed.

Value

Loading the data creates v.rapeseed.vcf, v.rapeseed.phe, and v.rapeseed.map, as described in the format section.

Source

Data provided by Rod Snowdon (University of Giessen).

---

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

Compacts, 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 *Compacts 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 STvcf/SelectionTools name convention used by the converters: identifiers are non-empty and contain no whitespace or control characters. Ordinary punctuation is permitted. The tropical-maize marker IDs happen to be 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` *Writes 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", pedigree = TRUE)
```

Arguments

pedigree	Logical; controls writing or reading quoted SelectionToolsPedigree records in the VCF header.
vcffile	Character string giving the output VCF file. If the name ends in .gz, the compiled 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

Full simulated pedigrees are stored in quoted SelectionToolsPedigree header records within the VCF, including compressed files. Sample names retain stable individual identifiers and may contain long pedigree-based names. Without these records, imported individuals are founders whose pedigree equals their sample name. Incomplete, duplicate or inconsistent pedigree records cause import to fail before replacing the destination. pedigree=FALSE omits these records on output and ignores them on input. Fitted prediction state is retained by the in-memory STvcf representation, rather than plain VCF.

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` *Writes version information in the package-specific format*

Description

Writes version information in the package-specific format.

Usage

```
write.version.2()
```

Value

An invisible `list` returned by the compiled 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**
gd.dummy.populations, 38
gd.maize.aflp, 41
gd.maize.lines, 42
gd.maize.populations, 42
gd.salad.aflp, 44
lib00.map1, 119
lib00.map1a, 120
lib00.map2, 120
masbc.input, 144
v-dentmaize-vcf, 407
v-dentmaize2-vcf, 408
v-flintmaize-vcf, 409
v-flintmaize2-vcf, 410
v-hybmaize-phe, 412
v-hybmaize2-phe, 413
v-lmm-climate, 414
v-lmm-corn, 414
v-lmm-data, 415
v-lmm-fertilizer-regression, 421
v-lmm-fertilizer-replicated, 421
v-lmm-halfdiallel, 422
v-lmm-iris, 423
v-lmm-iris-two, 424
v-lmm-machines, 425
v-lmm-mtcars, 426
v-lmm-rats, 427
v-lmm-tomato, 428
v-lmm-tropmaize, 428
v-lmm-tropmaize-lattice, 430
v-rapeseed, 431
v-tropmaize-map, 432
v-tropmaize-phe, 433
v-tropmaize-pop, 433
v-tropmaize-vcf, 434
∗ **hplot**
st.LDheatmap, 262
∗ **models**
influence.st.lmm, 114
st.lmm.compare, 317
st.lmm.components, 319
st.lmm.diagnostics, 324
st.lmm.model, 326
st.lmm.simulate, 333
.st_lmm_emm_DF *(st.lmm)*, 267
[.st.lmm.result *(st.lmm)*, 267
[[.st.lmm.result *(st.lmm)*, 267

AIC.st.lmm *(st.lmm)*, 267
alfalfa *(v-lmm-data)*, 415
anova, *319*
anova.st.lmm *(st.lmm)*, 267
append.population, 10
as.data.frame.st.gwas *(st.gwas)*, 249
as.data.frame.st.lmm.result *(st.lmm)*, 267
as.data.frame.st.lsm *(st.lsm)*, 342
as.data.frame.st.predict *(st.predict)*, 373
as.list.st.lmm.result *(st.lmm)*, 267
as.matrix.st.lmm.result *(st.lmm)*, 267
as.vector.st.lmm.result *(st.lmm)*, 267

barley.lattice *(v-lmm-data)*, 415
barley.met *(v-lmm-data)*, 415
barley.seven.traits *(v-lmm-data)*, 415
be.quiet, 11
BIC.st.lmm *(st.lmm)*, 267

chickens *(v-lmm-data)*, 415
climate *(v-lmm-climate)*, 414
coef.st.lmm *(st.lmm)*, 267
coef.st.lmm.by *(st.lmm.by)*, 315
coef.st.lsm *(st.lsm)*, 342
coef.st.predict *(st.predict)*, 373
compare.gwas.fits *(st.gwas)*, 249
concat.population, 11
confint, *330*
confint.st.lmm *(st.lmm)*, 267

confint.st.lmm.bootstrap
(st.lmm.bootstrap), 311
confint.st.lmm.by (st.lmm.by), 315
confint.st.lmm.profile
(st.lmm.profile), 327
confint.st.lsm (st.lsm), 342
confint.st.predict (st.predict), 373
cooks.distance.st.lmm
(influence.st.lmm), 114
copy.population, 12
corn (v-lmm-corn), 414
cotton (v-lmm-data), 415
cross, 12

data.params, 15
define.effects, 15
define.effects.df, 16
define.genome, 17, *379*
define.map, 18
deprecated, 20
df.residual.st.lmm (st.lmm), 267
dfbeta.st.lmm (influence.st.lmm), 114
dfbetas.st.lmm (influence.st.lmm), 114
dh, 20
divide.population, 21
drop1, *319*
drop1.st.lmm (st.lmm.compare), 317

effect.weight.set.all, 22
effmap.remove, 22
effmap.remove.all, 23
emm_basis.st.lmm (st.lmm), 267
evaluate.all.loci, 23
evaluate.allele, 24, *27*
evaluate.allele.freq, *24*, 25, *26*
evaluate.allele.freq2, *25*, 26, *27*, *168*
evaluate.allele2, *24*, 27
evaluate.genome, 28
evaluate.genotype, 29
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
extend.marker.map (linkage.map.get), 121

fertilizer (v-lmm-data), 415
fertilizer.regression
(v-lmm-fertilizer-regression),
421
fertilizer.replicated
(v-lmm-fertilizer-replicated),
421
fitted.st.lmm (st.lmm), 267
formula.st.lmm (st.lmm), 267

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, *206*
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, *364*
gs.cross.info.gd, 70
gs.cross.validation, 71
gs.esteff.lsq, 72
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, *80*
gs.get.Z, 79, *244*
gs.info, 80
gs.lambda.aov, 81
gs.lambda.const, 81
gs.lambda.emstep, 82
gs.lambda.hsq, 82
gs.lambda.reg, 83
gs.lambda.rmla, 84
gs.lambda.rmla.02, 84
gs.lambda.rmlc, 85
gs.lambda.rmlr, 86
gs.lambda.rmlv, 86
gs.lambda.rrblup, 87
gs.make.A, *80*, 88, *244*, *429*, *431*
gs.mme.coeff, 88
gs.mme.invcoeff, 89
gs.mme.restcoeff, 89
gs.mme.restrhs, 90
gs.mme.rhs, 90
gs.mme.solve, 91
gs.mmet.coeff, 92
gs.mmet.coeff.3, 92
gs.mmet.rhs, 93
gs.mmet.solve, 93
gs.neg.effall, 94
gs.plot.effects, 94
gs.plot.model.fit, 95
gs.plot.validation, 96
gs.pos.effall, 97
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, 101
gs.set.lambda, 102
gs.set.num.threads, 102, *393*
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

halfdiallel *(*v-lmm-halfdiallel*)*, 422
hatvalues.st.lmm *(*st.lmm.diagnostics*)*, 324
homozygote, 111

il.eval.library, 111
il.eval.lines, 112
il.ideal.library, 112
il.overlapping.library, 113
influence.st.lmm, 114, *326*
info, 117
info.cat, 118
init.population, 118
iris.manova *(*v-lmm-iris*)*, 423
iris.two *(*v-lmm-iris-two*)*, 424

lib00.map1, 119
lib00.map1a, 120

lib00.map2, 120
linkage.drag, 121
linkage.map.get, 121
linkage.map.load, 122
linkage.map.save, 123
list.effects, 124
list.populations, 124
load.effmap, 125
load.internal.effmap, 126
load.linkage.map, 126
logLik.st.lmm *(st.lmm)*, 267

mab.compare, 127, *134*, *142–144*
mab.df, 134
mab.Examples, *134*, 134, *135*, *137*, *142, 143*
mab.Input.files, *134*, 135, *142*, *144*
mab.load.data, *134*, 136, *142–144*
mab.save.inds, 137
mab.simulate, *134, 135*, *137*, 138, *143, 144*
mab.tabulate, *134*, *137*, *142*, 142, *144*
machines *(v-lmm-machines)*, 425
maize.aflp *(gd.maize.aflp)*, 41
maize.counties *(v-lmm-data)*, 415
maize.met *(v-lmm-data)*, 415
masbc.input, 144
material.groups *(v-lmm-data)*, 415
Math.st.lmm.result *(st.lmm)*, 267
met.missing *(v-lmm-data)*, 415
met.seven.traits *(v-lmm-data)*, 415
met.unbalanced *(v-lmm-data)*, 415
model.frame, *327*
model.frame.st.lmm *(st.lmm.model)*, 326
model.matrix.st.lmm *(st.lmm)*, 267
modelparm.st.lmm *(st.lmm)*, 267
mtcars *(v-lmm-mtcars)*, 426

nobs.st.lmm *(st.lmm)*, 267

oats *(v-lmm-data)*, 415
Ops.st.lmm.result *(st.lmm)*, 267
optim, *333*
optimize.population, 144

pairs.st.lsm *(st.lsm)*, 342
peas *(v-lmm-data)*, 415
ph.linkage.map.create, 145
phenotype.population, 146
plabsim, 147
plabsim.init, 147
plabsim.R.date, 147
plabsim.R.version, 148
plabsim.version, 148
plant, 148
plot.st.gwas *(st.gwas)*, 249
plot.st.lmm *(st.lmm)*, 267
plot.st.lmm.influence *(influence.st.lmm)*, 114
plot.st.lmm.profile *(st.lmm.profile)*, 327
population.append, *10*, 149
population.concat, 150
population.copy, 151
population.cross *(cross)*, 12
population.divide, 151
population.exist, 152
population.individual.remove, 153
population.info.get, 153
population.info.set, 154
population.list, 155
population.marker.map *(linkage.map.get)*, 121
population.matrix.load, 155
population.matrix.save, 156
population.name.swap, 156
population.optimize, 157
population.pedigree, *247*
population.pedigree *(population.info.get)*, 153
population.plabsim.load, 157
population.plabsim.save, 158
population.remove, 158
population.remove.all, 159
population.rename, 160
population.resize, 160
population.sample, 161, *178*
population.size.get, 162
population.sort, 162
population.swap.name, 163
population.transfer, 164
predict.st.lmm *(st.lmm)*, 267
print.propsd *(propsd)*, 164
print.st.cross.allocation *(st.cross.allocate)*, 202
print.st.cross.frontier *(st.cross.plan)*, 214
print.st.cross.plan *(st.cross.plan)*, 214
print.st.cross.population

(st.cross.evaluate), 208
print.st.data.set.list
(st.data.set.list), 227
print.st.gwas (st.gwas), 249
print.st.hybrid.data.set.list
(st.hybrid.data.set), 256
print.st.lmm (st.lmm), 267
print.st.lmm.allfit (st.lmm.allfit), 309
print.st.lmm.bootstrap
(st.lmm.bootstrap), 311
print.st.lmm.by (st.lmm.by), 315
print.st.lmm.influence
(influence.st.lmm), 114
print.st.lmm.profile (st.lmm.profile),
327
print.st.lsm (st.lsm), 342
print.st.ocs (st.ocs), 358
print.st.oma (st.oma), 361
print.st.predict (st.predict), 373
print.st.switch.error
(st.switch.error), 402
print.summary.st.lmm.by (st.lmm.by), 315
print.summary.st.lsm (st.lsm), 342
print.summary.st.predict (st.predict),
373
profile.st.lmm (st.lmm.profile), 327
propsd, 164, *208*, *214*

rapeseed (v-lmm-data), 415
rats (v-lmm-rats), 427
read.vcf, 166
recover_data.st.lmm (st.lmm), 267
remove.all.populations, 168
remove.effmaps, 169
remove.evaluate.population, 169
remove.genotype.population, 170
remove.map, 170
remove.population, 171
rename.population, 171
reset.all, 172
reset.mdp, 173
residuals.st.lmm (st.lmm), 267
resize.population, 173
resources, 174
return.population, 174
rng.choose, 175
rng.info, 175
rng.init, 176
rng.list, 177
rstandard.st.lmm (st.lmm.diagnostics),
324
rstudent.st.lmm, *326*
rstudent.st.lmm (influence.st.lmm), 114

sample.population, 177
save.linkage.map, 178
sdev, 179
sel.target.define, 179
select.all.best, 180
select.all.best.intern, 181
select.genotypes, 182
select.n.best, 183
select.n.best.segments, 184
set.co.freq, 185
set.eff.weight, 185
set.genome.par, *18*, 186
set.info.level, 187
set.mdp, 187
set.NoLociInit, 188
set.population.gvalue, 188
set.population.info, 189
show.phase, 190
simulate, *337*
simulate.st.lmm, *253*, *312*
simulate.st.lmm (st.lmm.simulate), 333
single.cross, 190
sm.start.timer, 191
sm.stop.timer, 191
soils (v-lmm-data), 415
soybean (v-lmm-data), 415
splitdt, 192
ssd.mating, 192
st.calc.ld, 193
st.calc.ld.2, 196
st.calc.q, 196
st.calc.rf, 197
st.calculate.enrichment, 198
st.chrom.stats, 200
st.compact.data.set, 200, *258*
st.copy.marker.data, 201, *388*
st.cross.allocate, 202, *208*, *214*
st.cross.constraints, *203*, 206, *209*, *217*
st.cross.evaluate, *203*, *205*, *208*, 208, *217*,
220, *224*
st.cross.frontier, *214*, *224*
st.cross.frontier (st.cross.plan), 214
st.cross.future, *204*, *210*, *214*, *217*, *224*
st.cross.future (st.cross.evaluate), 208

st.cross.plan, *206*, *208*, *214*, 214, *364*
st.cross.plan.execute, 224
st.cross.progeny, *206*, *212*, *214*, *217, 218*, *221*, *224*
st.cross.progeny *(*st.cross.plan*)*, 214
st.data.combine, 226
st.data.set.list, 227, *258*
st.data.set.remove, 229
st.data.set.remove.all, 230
st.datadir, 230
st.dataframe.to.STvcf, 231, *341*
st.dd, 233
st.def.hblocks, 234
st.destroy.phase, 235
st.detect.core, 236
st.diallel.cross, 238, *275*
st.generate.phenotypes *(*st.load.performance.data*)*, 337
st.genetic.distances, 239
st.genetic.distances.02, 242
st.genetic.distances.fct, 242
st.get.A, 243, *258*, *309*, *408*, *410*
st.get.hybrid.performance.data *(*st.hybrid.data.set*)*, 256
st.get.info.level, 244
st.get.map, 245, *341*
st.get.num.threads, 245, *393*
st.get.simpop, 247
st.get.simpop.perfdata, 248
st.get.simpop2, 248
st.gwas, 249
st.hybrid.data.set, 256, *272*, *413*
st.id, 258
st.indir, 258
st.info, 259
st.keep.intersection.markers, 259
st.kernel.structure, 260, *304*
st.LDheatmap, 262
st.LDplot.ld, 266
st.LDplot.map, 266
st.lmm, *117*, *238*, *251*, *258*, *261*, 267, *311*, *315–317*, *319*, *324*, *326, 327*, *330–333*, *336, 337*, *353, 354*, *377*, *413–415*, *420, 421*, *424, 425*, *427–429*, *431*
st.lmm.allfit, 309
st.lmm.bootstrap, *311*, 311
st.lmm.by, 315
st.lmm.compare, *271*, *275*, *281*, 317
st.lmm.components, *117*, *257*, *272*, *317*, 319, *326*, *333*, *337*
st.lmm.diagnostics, *117*, *270*, *273*, 324
st.lmm.model, *281, 282*, *319*, 326
st.lmm.objective, *311*, *330*
st.lmm.objective *(*st.lmm.refit*)*, 331
st.lmm.pca *(*st.lmm.components*)*, 319
st.lmm.profile, *269*, *309*, *311*, *315*, 327
st.lmm.refit, *117*, *311, 312*, *315*, *317*, *330*, 331, *337*
st.lmm.simulate, *315*, 333
st.lmm.singular *(*st.lmm.components*)*, 319
st.load.performance.data, 337
st.load.vcf.data, *244*, 339, *382*, *408–411*
st.lsm, *287*, *309*, *324*, 342, *354*, *374*, *377*, *414, 415*, *420, 421*, *424, 425*, *427–429*, *431*
st.lsm-hypotheses, 353
st.mark.alleles, 354
st.marker.data.statistics, 355, *382*
st.marker.results, *288*
st.marker.results *(*st.predict*)*, 373
st.markerdata.to.STvcf, *341*, 356
st.ocs, 358, *364*
st.od, 360
st.oma, 361
st.options, *206*, *257*, *310*, *328*, *364*
st.options *(*st.lmm*)*, 267
st.outdir, 364
st.phase, 364
st.plot.corr, 368
st.plot.corr.l, 368
st.plot.gene.diversity, *356*, 369
st.plot.genetic.distances *(*st.genetic.distances*)*, 239
st.plot.ggt, 370
st.plot.ggt.src, 372
st.plot.pcoa *(*st.genetic.distances*)*, 239
st.predict, *238*, *276*, *288*, *309*, *324*, 373
st.read.map, 377, *379*
st.read.map.df, 379
st.read.marker.data, 380, *382*
st.read.marker.data.df, *244*, *379*, 381
st.read.performance.data, *339*, 383
st.recode.hbc, 384
st.recode.hil, 385
st.recode.ref, 386

st.recode.ref.2, 387
st.repack.data.set (st.compact.data.set), 200
st.reset, 387
st.restrict.marker.data, *356*, 388, *408*, *410*
st.return.performance.data, 389
st.select.enrichment (st.calculate.enrichment), 198
st.select.phen, 390
st.set.hblocks, 391
st.set.hybrid.performance.data (st.hybrid.data.set), 256
st.set.info.level, 391
st.set.matrix.ops, 392
st.set.num.threads, *115*, *246*, *253*, *310*, *313*, *316*, *329*, 392
st.set.openblas.threads, 393
st.set.sim.ef, 394
st.set.sim.gp, 396
st.set.sim.mp, 397
st.set.sim.pp, 397
st.set.simpop, 398, *409*, *411*
st.start.timer, 400
st.stop.timer, 400
st.STvcf.to.dataframe, *341*, 401
st.switch.error, 402
st.write.map, 404
st.write.marker.data, 404
sugar (v-lmm-data), 415
summarize.gvalue, 406
summarize.gwas.fit (st.gwas), 249
summarize.gwas.simulation (st.gwas), 249
summary.st.lmm (st.lmm), 267
summary.st.lmm.allfit (st.lmm.allfit), 309
summary.st.lmm.bootstrap (st.lmm.bootstrap), 311
summary.st.lmm.by (st.lmm.by), 315
summary.st.lmm.influence (influence.st.lmm), 114
Summary.st.lmm.result (st.lmm), 267
summary.st.lsm (st.lsm), 342
summary.st.predict (st.predict), 373
swap.population.name, 406

t.st.lmm.result (st.lmm), 267
talk.to.me, 407
terms.st.lmm (st.lmm.model), 326
tomato (v-lmm-tomato), 428
tropmaize (v-lmm-tropmaize), 428
tropmaize.lattice (v-lmm-tropmaize-lattice), 430

update, *327*, *332*, *333*
update.formula, *326*
update.st.lmm (st.lmm.model), 326

v-dentmaize-vcf, 407
v-dentmaize2-vcf, 408
v-flintmaize-vcf, 409
v-flintmaize2-vcf, 410
v-hybmaize-phe, 412
v-hybmaize2-phe, 413
v-lmm-alfalfa (v-lmm-data), 415
v-lmm-barley-lattice (v-lmm-data), 415
v-lmm-barley-met (v-lmm-data), 415
v-lmm-barley-seven-traits (v-lmm-data), 415
v-lmm-chickens (v-lmm-data), 415
v-lmm-climate, 414
v-lmm-corn, 414
v-lmm-cotton (v-lmm-data), 415
v-lmm-data, 415
v-lmm-fertilizer (v-lmm-data), 415
v-lmm-fertilizer-regression, 421
v-lmm-fertilizer-replicated, 421
v-lmm-halfdiallel, 422
v-lmm-iris, 423
v-lmm-iris-two, 424
v-lmm-machines, 425
v-lmm-maize-counties (v-lmm-data), 415
v-lmm-maize-met (v-lmm-data), 415
v-lmm-material-groups (v-lmm-data), 415
v-lmm-met-missing (v-lmm-data), 415
v-lmm-met-seven-traits (v-lmm-data), 415
v-lmm-met-unbalanced (v-lmm-data), 415
v-lmm-mtcars, 426
v-lmm-oats (v-lmm-data), 415
v-lmm-peas (v-lmm-data), 415
v-lmm-rapeseed (v-lmm-data), 415
v-lmm-rapeseed-blocks (v-lmm-data), 415
v-lmm-rats, 427
v-lmm-soils (v-lmm-data), 415
v-lmm-soybean (v-lmm-data), 415
v-lmm-soybean-blocks (v-lmm-data), 415
v-lmm-soybean-met (v-lmm-data), 415
v-lmm-sugar (v-lmm-data), 415

v-lmm-tomato, 428
v-lmm-tropmaize, 428
v-lmm-tropmaize-lattice, 430
v-rapeseed, 431
v-tropmaize-map, 432
v-tropmaize-phe, 433
v-tropmaize-pop, 433
v-tropmaize-vcf, 434
v.dentmaize.vcf *(v-dentmaize-vcf)*, 407
v.dentmaize2.vcf *(v-dentmaize2-vcf)*, 408
v.flintmaize.vcf *(v-flintmaize-vcf)*, 409
v.flintmaize2.vcf *(v-flintmaize2-vcf)*, 410
v.hybmaize.phe *(v-hybmaize-phe)*, 412
v.hybmaize2.phe *(v-hybmaize2-phe)*, 413
v.rapeseed.map *(v-rapeseed)*, 431
v.rapeseed.phe *(v-rapeseed)*, 431
v.rapeseed.vcf *(v-rapeseed)*, 431
v.tropmaize.map *(v-tropmaize-map)*, 432
v.tropmaize.phe *(v-tropmaize-phe)*, 433
v.tropmaize.pop *(v-tropmaize-pop)*, 433
v.tropmaize.vcf *(v-tropmaize-vcf)*, 434
vcov.st.lmm *(st.lmm)*, 267
vcov.st.lsm *(st.lsm)*, 342
vcov.st.predict *(st.predict)*, 373

weights.st.lmm *(st.lmm.model)*, 326
write.vcf, 435
write.version.2, 437