Package ‘qtl2’
April 22, 2023

Version  0.32
Date     2023-04-21
Title    Quantitative Trait Locus Mapping in Experimental Crosses
Description Provides a set of tools to perform quantitative trait locus (QTL) analysis in experimental crosses. It is a reimplemention of the 'R/qtl' package to better handle high-dimensional data and complex cross designs. Broman et al. (2019) <doi:10.1534/genetics.118.301595>.

Author   Karl W Broman [aut, cre] (<https://orcid.org/0000-0002-4914-6671>), R Core Team [ctb]
Maintainer Karl W Broman <broman@wisc.edu>

Copyright Code for Brent's method for univariate function optimization was taken from R 3.2.2 (Copyright 1995, 1996 Robert Gentleman and Ross Ihaka, Copyright 2003-2004 The R Foundation, Copyright 1998-2014 The R Core Team).

Depends  R (>= 3.1.0)
Imports  Rcpp (>= 1.0.7), yaml (>= 2.1.13), jsonlite (>= 0.9.17), data.table (>= 1.10.4-3), parallel, stats, utils, graphics, grDevices, RSQLite
Suggests testthat, devtools, roxygen2, vdiffr, qtl
License  GPL-3
BugReports https://github.com/rqtl/qtl2/issues
LazyData true
Encoding UTF-8
ByteCompile true
LinkingTo Rcpp, RcppEigen
RoxygenNote  7.2.3
NeedsCompilation yes
Repository CRAN
Date/Publication 2023-04-22 00:52:43 UTC
R topics documented:

- add_threshold .................................................. 4
- basic_summaries ............................................... 5
- batch_cols ....................................................... 7
- batch_vec ......................................................... 8
- bayes_int ......................................................... 9
- calc_entropy ..................................................... 10
- calc_errorlod ................................................... 11
- calc_genoprob .................................................... 12
- calc_geno_freq ................................................ 14
- calc_grid ......................................................... 15
- calc_het ......................................................... 16
- calc_kinship ..................................................... 16
- calc_raw_founder_maf ......................................... 18
- calc_raw_geno_freq ........................................... 19
- calc_raw_het .................................................... 20
- calc_raw_maf ................................................... 20
- calc_sdp ......................................................... 21
- cbind.calc_genoprob ........................................... 22
- cbind.scan1 ...................................................... 23
- cbind.scan1perm ................................................. 24
- cbind.sim_geno .................................................. 25
- cbind.viterbi .................................................... 26
- cbind_expand .................................................... 26
- CCcolors ........................................................ 27
- check_cross2 ..................................................... 28
- chisq_colpairs .................................................. 28
- chr_lengths ..................................................... 29
- clean ............................................................ 30
- clean_genoprob ................................................ 30
- clean_scan1 ..................................................... 32
- compare_geno ................................................... 33
- compare_genoprob .............................................. 34
- compare_maps ................................................... 35
- convert2cross2 ................................................ 36
- count_xo ........................................................ 37
- create_gene_query_func ........................................ 38
- create_snpinfo ................................................ 39
- create_variant_query_func .................................... 40
- decomp_kinship ............................................... 42
- drop_markers ................................................... 43
- drop_nullmarkers .............................................. 43
- est_herit ........................................................ 44
- est_map .......................................................... 46
- find_ibd_segments .............................................. 47
- find_index_snp ................................................ 49
- find_map_gaps .................................................. 50
R topics documented:

find_marker .......................................................... 51
find_markerpos ......................................................... 52
find_peaks ............................................................... 53
fit1 ................................................................. 55
genoprob_toalleleprob .................................................. 58
genoprob_to_snpprob .................................................... 59
get_common_ids ......................................................... 61
get_x_covar ............................................................. 62
guess_phase ............................................................ 62
index_snps ............................................................... 63
insert_pseudomarkers .................................................. 65
interp_genoprob ........................................................ 66
interp_map ............................................................... 67
invert_sdp ............................................................... 68
locate_xo ................................................................. 69
lod_int ................................................................. 70
map_to_grid ........................................................... 71
mat2strata .............................................................. 72
maxlod ................................................................. 73
maxmarg ................................................................. 74
max_compare_geno ....................................................... 75
max_scan1 ............................................................... 76
n_missing ............................................................... 77
plot_coef ............................................................... 78
plot_compare_geno ...................................................... 81
plot_genes ............................................................. 81
plot_genoprob .......................................................... 83
plot_genoprobcomp ..................................................... 85
plot_lodpeaks .......................................................... 87
plot_onegeno ........................................................... 88
plot_peaks ............................................................. 90
plot_pgx ............................................................... 92
plot_scan1 ............................................................. 94
plot_sdp ............................................................... 95
plot_snpasso ........................................................... 96
predict_snpgeno ........................................................ 99
print.cross2 .......................................................... 100
print.summary.scan1perm ............................................. 100
probs_to_grid ........................................................ 101
pull_genoprobint ...................................................... 102
pull_genoprobpos ...................................................... 103
pull_markers .......................................................... 104
qtl2version ............................................................ 105
rbind.calc_genoprob ................................................... 105
rbind.scan1 ........................................................... 106
rbind.scan1perm ....................................................... 107
rbind.sim_geno ........................................................ 108
rbind.viterbi .......................................................... 109
add_threshold

Add thresholds to genome scan plot

Description

Draw line segments at significance thresholds for a genome scan plot

Usage

add_threshold(map, thresholdA, thresholdX = NULL, chr = NULL, gap = NULL, ...)
Arguments

- **map**: Marker map used in the genome scan plot
- **thresholdA**: Autosomal threshold. Numeric or a list. (If a list, the "A" component is taken to be `thresholdA` and the "X" component is taken to be `thresholdX`.)
- **thresholdX**: X chromosome threshold (if missing, assumed to be the same as `thresholdA`)
- **chr**: Chromosomes that were included in the plot
- **gap**: Gap between chromosomes in the plot. Default is 1% of the total genome length.
- **...**: Additional arguments passed to `graphics::segments()`

Value

None.

Examples

```r
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

map <- insert_pseudomarkers(iron$gmap, step=5)
probs <- calc_genoprob(iron, map, error_prob=0.002)
Xcovar <- get_x_covar(iron)
out <- scan1(probs, iron$pheno[,1], Xcovar=Xcovar)
# run just 3 permutations, as a fast illustration
operm <- scan1perm(probs, iron$pheno[,1], addcovar=Xcovar,
   n_perm=3, perm_Xsp=TRUE, chr_lengths=chr_lengths(map))

plot(out, map)
add_threshold(map, summary(operm), col="violetred", lty=2)
```

basic_summaries

Basic summaries of a cross2 object

Description

Basic summaries of a cross2 object.

Usage

- `n_ind(cross2)`
- `n_ind_geno(cross2)`
- `n_ind_pheno(cross2)`
- `n_ind_covar(cross2)`
- `n_ind_gnp(cross2)`
\begin{verbatim}
ind_ids(cross2)
ind_ids_geno(cross2)
ind_ids_pheno(cross2)
ind_ids_covar(cross2)
ind_ids_gnp(cross2)
n_chr(cross2)
n_founders(cross2)
founders(cross2)
chr_names(cross2)
tot_mar(cross2)
n_mar(cross2)
marker_names(cross2)
n_pheno(cross2)
pheno_names(cross2)
n_covar(cross2)
covar_names(cross2)
n_phenocovar(cross2)
phenocovar_names(cross2)
\end{verbatim}

**Arguments**

- **cross2** An object of class "cross2", as output by read_cross2(). For details, see the R/qtl2 developer guide.

**Value**

Variously a number, vector of numbers, or vector of character strings.

**Functions**

- n_ind(): Number of individuals (either genotyped or phenotyped)
• n_ind_geno(): Number of genotyped individuals
• n_ind_pheno(): Number of phenotyped individuals
• n_ind_covar(): Number of individuals with covariate data
• n_ind_gnp(): Number of individuals with both genotype and phenotype data
• ind_ids(): IDs of individuals (either genotyped or phenotyped)
• ind_ids_geno(): IDs of genotyped individuals
• ind_ids_pheno(): IDs of phenotyped individuals
• ind_ids_covar(): IDs of individuals with covariate data
• ind_ids_gnp(): IDs of individuals with both genotype and phenotype data
• n_chr(): Number of chromosomes
• n_founders(): Number of founder strains
• founders(): Names of founder strains
• chr_names(): Chromosome names
• tot_mar(): Total number of markers
• n_mar(): Number of markers on each chromosome
• marker_names(): Marker names
• n_pheno(): Number of phenotypes
• pheno_names(): Phenotype names
• n_covar(): Number of covariates
• covar_names(): Covariate names
• n_phenocovar(): Number of phenotype covariates
• phenocovar_names(): Phenotype covariate names

See Also

summary.cross2()

---

**Description**

Identify batches of columns of a matrix that have the same pattern of missing values.

**Usage**

```r
batch_cols(mat, max_batch = NULL)
```

**Arguments**

- `mat`: A numeric matrix
- `max_batch`: Maximum batch size
Value
A list containing the batches, each with two components: cols containing numeric indices of the columns in the corresponding batch, and omit containing a vector of row indices that have missing values in this batch.

See Also
batch_vec()

Examples
```r
x <- rbind(c( 1, 2, 3, 13, 16),
           c( 4, 5, 6, 14, 17),
           c( 7, NA, 8, NA, 18),
           c(NA, NA, NA, NA, 19),
           c(10, 11, 12, 15, 20))
batch_cols(x)
```

---

**batch_vec**

*Split vector into batches*

Description
Split a vector into batches, each no longer than batch_size and creating at least n_cores batches, for use in parallel calculations.

Usage
`batch_vec(vec, batch_size = NULL, n_cores = 1)`

Arguments
- `vec` A vector to be split into batches
- `batch_size` Maximum size for each batch
- `n_cores` Number of compute cores, to be used as a minimum number of batches.

Value
A list of vectors, each no longer than batch_size, and with at least n_cores components.

See Also
batch_cols()

Examples
```r
vec_split <- batch_vec(1:304, 50, 8)
vec_split2 <- batch_vec(1:304, 50)
```
**bayes_int**

*Calculate Bayes credible intervals*

**Description**

Calculate Bayes credible intervals for a single LOD curve on a single chromosome, with the ability to identify intervals for multiple LOD peaks.

**Usage**

```r
bayes_int(
    scan1_output,
    map,
    chr = NULL,
    lodcolumn = 1,
    threshold = 0,
    peakdrop = Inf,
    prob = 0.95,
    expand2markers = TRUE
)
```

**Arguments**

- `scan1_output`: An object of class "scan1" as returned by `scan1()`.
- `map`: A list of vectors of marker positions, as produced by `insert_pseudomarkers()`.
- `chr`: Chromosome ID to consider (must be a single value).
- `lodcolumn`: LOD score column to consider (must be a single value).
- `threshold`: Minimum LOD score for a peak.
- `peakdrop`: Amount that the LOD score must drop between peaks, if multiple peaks are to be defined on a chromosome.
- `prob`: Nominal coverage for the interval.
- `expand2markers`: If TRUE, QTL intervals are expanded so that their endpoints are at genetic markers.

**Details**

We identify a set of peaks defined as local maxima that exceed the specified `threshold`, with the requirement that the LOD score must have dropped by at least `peakdrop` below the lowest of any two adjacent peaks.

At a given peak, if there are ties, with multiple positions jointly achieving the maximum LOD score, we take the average of these positions as the location of the peak.

The default is to use `threshold=0`, `peakdrop=Inf`, and `prob=0.95`. We then return results a single peak, no matter the maximum LOD score, and give a 95% Bayes credible interval.
Description

For each individual at each genomic position, calculate the entropy of the genotype probability distribution, as a quantitative summary of the amount of missing information.

Usage

calc_entropy(probs, quiet = TRUE, cores = 1)
Arguments

probs Genotype probabilities, as calculated from `calc_genoprob()`.
quiet IF FALSE, print progress messages.
cores Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`.) Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.

Details

We calculate \(-\sum(p \log_2 p)\), where we take 0 \(\log 0 = 0\).

Value

A list of matrices (each matrix is a chromosome and is arranged as individuals x markers).

Examples

```r
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))

probs <- calc_genoprob(grav2, error_prob=0.002)
e <- calc_entropy(probs)
e <- do.call("cbind", e) # combine chromosomes into one big matrix

# summarize by individual
mean_ind <- rowMeans(e)

# summarize by marker
mean_marker <- colMeans(e)
```

Description

Use the genotype probabilities calculated with `calc_genoprob()` to calculate genotyping error LOD scores, to help identify potential genotyping errors (and problem markers and/or individuals).

Usage

```r
calc_errorlod(cross, probs, quiet = TRUE, cores = 1)
```

Arguments

cross Object of class "cross2". For details, see the R/qtl2 developer guide.
probs Genotype probabilities as calculated from `calc_genoprob()`.
quiet If FALSE, print progress messages.
cores Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`.) Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.
Details

Let $O_k$ denote the observed marker genotype at position $k$, and $g_k$ denote the corresponding true underlying genotype.

Following Lincoln and Lander (1992), we calculate $\text{LOD} = \log_{10}[Pr(O_k|g_k = O_k)/Pr(O_k|g_k \neq O_K)]$

Value

A list of matrices of genotyping error LOD scores. Each matrix corresponds to a chromosome and is arranged as individuals x markers.

References


See Also

calc_genoprob()

Examples

```r
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
probs <- calc_genoprob(iron, error_prob=0.002, map_function="c-f")
errorlod <- calc_errorlod(iron, probs)

# combine into one matrix
errorlod <- do.call("cbind", errorlod)
```

Description

Uses a hidden Markov model to calculate the probabilities of the true underlying genotypes given the observed multipoint marker data, with possible allowance for genotyping errors.

Usage

```r
calc_genoprob(
cross,
  map = NULL,
  error_prob = 0.0001,
  map_function = c("haldane", "kosambi", "c-f", "morgan"),
  lowmem = FALSE,
  quiet = TRUE,
  cores = 1
)```
Arguments

cross  Object of class "cross2". For details, see the R/qtl2 developer guide.
map    Genetic map of markers. May include pseudomarker locations (that is, locations that are not within the marker genotype data). If NULL, the genetic map in cross is used.
error_prob  Assumed genotyping error probability
map_function Character string indicating the map function to use to convert genetic distances to recombination fractions.
lowmem  If FALSE, split individuals into groups with common sex and crossinfo and then precalculate the transition matrices for a chromosome; potentially a lot faster but using more memory.
quiet  If FALSE, print progress messages.
cores  Number of CPU cores to use, for parallel calculations. (If 0, use parallel::detectCores.) Alternatively, this can be links to a set of cluster sockets, as produced by parallel::makeCluster.

Details

Let $O_k$ denote the observed marker genotype at position $k$, and $g_k$ denote the corresponding true underlying genotype.

We use the forward-backward equations to calculate $\alpha_{kv} = \log Pr(O_1, \ldots, O_k, g_k = v)$ and $\beta_{kv} = \log Pr(O_{k+1}, \ldots, O_n | g_k = v)$

We then obtain $Pr(g_k | O_1, \ldots, O_n) = \exp(\alpha_{kv} + \beta_{kv}) / s$ where $s = \sum_v \exp(\alpha_{kv} + \beta_{kv})$

Value

An object of class "calc_genoprob": a list of three-dimensional arrays of probabilities, individuals x genotypes x positions. (Note that the arrangement is different from R/qtl.) Also contains four attributes:

- crosstype - The cross type of the input cross.
- is_X_chr - Logical vector indicating whether chromosomes are to be treated as the X chromosome or not, from input cross.
- alleles - Vector of allele codes, from input cross.
- alleleprobs - Logical value (FALSE) that indicates whether the probabilities are compressed to allele probabilities, as from genoprob_to_alleleprob.

See Also

insert_pseudomarkers()

Examples

grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
gmap_w_pmar <- insert_pseudomarkers(grav2$gmap, step=1)
probs <- calc_genoprob(grav2, gmap_w_pmar, error_prob=0.002)
calc_gen_freq

Calculate genotype frequencies

Description
Calculate genotype frequencies, by individual or by marker

Usage

```r
calc_gen_freq(probs, by = c("individual", "marker"), omit_x = TRUE)
```

Arguments

- **probs**: List of arrays of genotype probabilities, as calculated by `calc_genoprob()`.
- **by**: Whether to summarize by individual or marker
- **omit_x**: If TRUE, results are just for the autosomes. If FALSE, results are a list of length two, containing the results for the autosomes and those for the X chromosome.

Value

If `omit_x`=TRUE, the result is a matrix of genotype frequencies; columns are genotypes and rows are either individuals or markers.

If necessary (that is, if `omit_x`=FALSE, the data include the X chromosome, and the set of genotypes on the X chromosome are different than on the autosomes), the result is a list with two components (for the autosomes and for the X chromosome), each being a matrix of genotype frequencies.

Examples

```r
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
p <- calc_genoprob(iron, err=0.002)

# genotype frequencies by marker
tab_g <- calc_gen_freq(p, "marker")

# allele frequencies by marker
ap <- genoprob_to_alleleprob(p)
tab_a <- calc_gen_freq(ap, "marker")
```
calc_grid

Calculate indicators of which marker/pseudomarker positions are along a fixed grid

Description

Construct vectors of logical indicators that indicate which positions correspond to locations along a grid.

Usage

calc_grid(map, step = 0, off_end = 0, tol = 0.01)

Arguments

map: A list of numeric vectors; each vector gives marker positions for a single chromosome.
step: Distance between pseudomarkers and markers; if step=0 no pseudomarkers are inserted.
off_end: Distance beyond terminal markers in which to insert pseudomarkers.
tol: Tolerance for determining whether a pseudomarker would duplicate a marker position.

Details

The function insert_pseudomarkers(), with stepwidth="fixed", will insert a grid of pseudomarkers, to a marker map. The present function gives a series of TRUE/FALSE vectors that indicate which positions fall on the grid. This is for use with probs_to_grid(), for reducing genotype probabilities, calculated with calc_genoprob(), to just the positions on the grid. The main value of this is to speed up genome scan computations in the case of very dense markers, by focusing on just a grid of positions rather than on all marker locations.

Value

A list of logical (TRUE/FALSE) vectors that indicate, for a marker/pseudomarker map created by insert_pseudomarkers() with step>0 and stepwidth="fixed", which positions correspond to the locations along the fixed grid.

See Also

insert_pseudomarkers(), probs_to_grid(), map_to_grid()

Examples

iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
gmap_w_pmar <- insert_pseudomarkers(iron$gmap, step=1)
grid <- calc_grid(iron$gmap, step=1)
calc_het  

**Calculate heterozygosities**

**Description**

Calculate heterozygosites, by individual or by marker

**Usage**

```
calc_het(probs, by = c("individual", "marker"), omit_x = TRUE, cores = 1)
```

**Arguments**

- `probs`: List of arrays of genotype probabilities, as calculated by `calc_genoprob()`.
- `by`: Whether to summarize by individual or marker
- `omit_x`: If TRUE, omit the X chromosome.
- `cores`: Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`. Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.

**Value**

The result is a vector of estimated heterozygosities

**Examples**

```
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
p <- calc_genoprob(iron, err=0.002)

# heterozygosities by individual
het_ind <- calc_het(p)

# heterozygosities by marker
het_mar <- calc_het(p, "marker")
```

calc_kinship  

**Calculate kinship matrix**

**Description**

Calculate genetic similarity among individuals (kinship matrix) from conditional genotype probabilities.
calc_kinship

Usage

calc_kinship(
  probs,
  type = c("overall", "loco", "chr"),
  omit_x = FALSE,
  use_allele_probs = TRUE,
  quiet = TRUE,
  cores = 1
)

Arguments

probs       Genotype probabilities, as calculated from calc_genoprob().
type        Indicates whether to calculate the overall kinship ("overall", using all chromosomes), the kinship matrix leaving out one chromosome at a time ("loco"), or the kinship matrix for each chromosome ("chr").
omit_x      If TRUE, only use the autosomes; ignored when type="chr".
use_allele_probs If TRUE, assess similarity with allele probabilities (that is, first run genoprob_to_alleleprob()); otherwise use the genotype probabilities.
quiet       IF FALSE, print progress messages.
cores       Number of CPU cores to use, for parallel calculations. (If 0, use parallel::detectCores().)

Alternatively, this can be links to a set of cluster sockets, as produced by parallel::makeCluster().

Details

If use_allele_probs=TRUE (the default), we first convert the genotype probabilities are converted to allele probabilities (using genoprob_to_alleleprob()). This is recommended, as then the result is twice the empirical kinship coefficient (e.g., the expected value for an intercross is 1/2; using genotype probabilities, the expected value is 3/8).

We then calculate \( \sum_{k,l} (P_{ilk}P_{jkl}) \) where \( k \) = position, \( l \) = allele, and \( i,j \) are two individuals.

For crosses with just two possible genotypes (e.g., backcross), we don’t convert to allele probabilities but just use the original genotype probabilities.

Value

If type="overall" (the default), a matrix of proportion of matching alleles. Otherwise a list with one matrix per chromosome.

Examples

grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
map <- insert_pseudomarkers(grav2$gmap, step=1)
probs <- calc_genoprob(grav2, map, error_prob=0.002)
K <- calc_kinship(probs)

# using only markers/pseudomarkers on the grid
grid <- calc_grid(grav2$gmap, step=1)  
probs_sub <- probs_to_grid(probs, grid)  
K_grid <- calc_kinship(probs_sub)

calc_raw_founder_maf  

**Description**

Calculate founder minor allele frequencies from raw SNP genotypes

**Usage**

calc_raw_founder_maf(cross, by = c("individual", "marker"))

**Arguments**

cross  
Object of class "cross2". For details, see the R/qtl2 developer guide.

by  
Indicates whether to summarize by founder strain ("individual") or by marker.

**Value**

A vector of minor allele frequencies, one for each founder strain or marker.

**See Also**

recode_snps(), calc_raw_maf()

**Examples**

```r
## Not run:
# load example data and calculate genotype probabilities
file <- paste0("https://raw.githubusercontent.com/rqtl/",
    "qtl2data/main/DOex/DOex.zip")
DOex <- read_cross2(file)
DOex_maf <- calc_raw_founder_maf(DOex)
## End(Not run)
```
calc_raw_geno_freq

Description

Calculate genotype frequencies from raw SNP genotypes, by individual or by marker

Usage

calc_raw_geno_freq(cross, by = c("individual", "marker"), cores = 1)

Arguments

cross Object of class "cross2". For details, see the R/qtl2 developer guide.
by Indicates whether to summarize by individual or by marker.
cores Number of CPU cores to use, for parallel calculations. (If 0, use parallel::detectCores.) Alternatively, this can be links to a set of cluster sockets, as produced by parallel::makeCluster.

Value

A matrix of genotypes frequencies with 3 columns (AA, AB, and BB) and with rows being either individuals or markers.

See Also

calc_raw_maf(), calc_raw_het(), recode_snps

Examples

## Not run:
# load example data and calculate genotype probabilities
file <- paste0("https://raw.githubusercontent.com/rqtl/",
  "qtl2data/main/DOex/DOex.zip")
DOex <- read_cross2(file)
gfreq <- calc_raw_geno_freq(DOex)
## End(Not run)
**calc_raw_maf**

*Calculate minor allele frequency from raw SNP genotypes*

**Description**
Calculate minor allele frequency from raw SNP genotypes, by individual or by marker

**Usage**
calc_raw_maf(cross, by = c("individual", "marker"))

**Arguments**
cross Object of class "cross2". For details, see the R/qtl2 developer guide.
by Indicates whether to summarize by founder strain ("individual") or by marker.

**Value**
A vector of frequencies, one for each individual or marker.

**See Also**
recode_snps(), calc_raw_maf(), calc_raw_founder_maf(), calc_raw_geno_freq()

**Examples**
```r
## Not run:
# load example data and calculate genotype probabilities
file <- paste0("https://raw.githubusercontent.com/rqtl/",
              "qtl2data/main/DOex/DOex.zip")
DOex <- read_cross2(file)
DOex_het <- calc_raw_het(DOex)
## End(Not run)
```

**calc_raw_het**

*Calculate estimated heterozygosity from raw SNP genotypes*

**Description**
Calculate estimated heterozygosity for each individual from raw SNP genotypes

**Usage**
calc_raw_het(cross, by = c("individual", "marker"))

**Arguments**
cross Object of class "cross2". For details, see the R/qtl2 developer guide.
by Indicates whether to summarize by founder strain ("individual") or by marker.

**Value**
A vector of heterozygosities, one for each individual or marker.

**See Also**
recode_snps(), calc_raw_maf(), calc_raw_founder_maf(), calc_raw_geno_freq()

**Examples**
```r
## Not run:
# load example data and calculate genotype probabilities
file <- paste0("https://raw.githubusercontent.com/rqtl/",
              "qtl2data/main/DOex/DOex.zip")
DOex <- read_cross2(file)
DOex_het <- calc_raw_het(DOex)
## End(Not run)
```
Arguments

cross  Object of class "cross2". For details, see the R/qtl2 developer guide.

by  Indicates whether to summarize by founder strain ("individual") or by marker.

Value

A vector of minor allele frequencies, one for each individual or marker.

See Also

recode_snps(), calc_raw_founder_maf(), calc_raw_het(), calc_raw_geno_freq()

Examples

## Not run:
# load example data and calculate genotype probabilities
file <- paste0("https://raw.githubusercontent.com/rqtl/",
   "qtl2data/main/DOex/DOex.zip")
DOex <- read_cross2(file)
DOex_maf <- calc_raw_maf(DOex)

## End(Not run)
Examples

```r
x <- rbind(m1=c(3, 1, 1, 1, 1, 1, 1, 1),
            m2=c(1, 3, 3, 1, 1, 1, 1, 1),
            m3=c(1, 1, 1, 1, 3, 3, 3, 3))
calc_sdp(x)
```

---

**cbind.calc_genoprob**  
*Join genotype probabilities for different chromosomes*

### Description

Join multiple genotype probability objects, as produced by `calc_genoprob()`, for the same set of individuals but different chromosomes.

### Usage

```r
## S3 method for class 'calc_genoprob'
cbind(...)  
```

### Arguments

- `...`  
  Genotype probability objects as produced by `calc_genoprob()`. Must have the same set of individuals.

### Value

An object of class "calc_genoprob", like the input; see `calc_genoprob()`.

### See Also

- `rbind.calc_genoprob()`

### Examples

```r
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
map <- insert_pseudomarkers(grav2$gmap, step=1)
probsA <- calc_genoprob(grav2[1:5,1:2], map, error_prob=0.002)
probsB <- calc_genoprob(grav2[1:5,3:4], map, error_prob=0.002)
probs <- cbind(probsA, probsB)
```
Join genome scan results for different phenotypes.

Description

Join multiple `scan1()` results for different phenotypes; must have the same map.

Usage

```r
## S3 method for class 'scan1'
cbind(...)  
```

Arguments

... Genome scan objects of class "scan1", as produced by `scan1()`. Must have the same map.

Details

If components `addcovar()`, `Xcovar`, `intcovar`, `weights` do not match between objects, we omit this information.

If `hsq` present but has differing numbers of rows, we omit this information.

Value

An object of class "scan1", like the inputs, but with the lod score columns from the inputs combined as multiple columns in a single object.

See Also

`rbind.scan1()`, `scan1()`

Examples

```r
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
map <- insert_pseudomarkers(grav2$gmap, step=1)
probs <- calc_genoprob(grav2, map, error_prob=0.002)
phe1 <- grav2$pheno[,1,drop=FALSE]
phe2 <- grav2$pheno[,2,drop=FALSE]
out1 <- scan1(probs, phe1) # phenotype 1
out2 <- scan1(probs, phe2) # phenotype 2
out <- cbind(out1, out2)
```
cbind.scan1perm

Combine columns from multiple scan1 perm permutation results

Description
Column-bind multiple scan1perm objects with the same numbers of rows.

Usage
## S3 method for class 'scan1perm'
cbind(...)

Arguments
... A set of permutation results from scan1perm() (objects of class "scan1perm". If different numbers of permutation replicates were used, those columns with fewer replicates are padded with missing values NA. However, if any include autosome/X chromosome-specific permutations, they must all be such.

Details
The aim of this function is to concatenate the results from multiple runs of a permutation test with scan1perm(), generally with different phenotypes and/or methods, to be used in parallel with rbind.scan1perm().

Value
The combined column-bound input, as an object of class "scan1perm"; see scan1perm().

See Also
rbind.scan1perm(), scan1perm(), scan1()

Examples
# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)

# permutations with genome scan (just 3 replicates, for illustration)
operm1 <- scan1perm(probs, pheno[,1,drop=FALSE], addcovar= covar, Xcovar=Xcovar, n_perm=3)
operm2 <- scan1perm(probs, pheno[,2,drop=FALSE], addcovar= covar, Xcovar=Xcovar, n_perm=3)

operm <- cbind(operm1, operm2)

---

### cbind.sim_geno

Join genotype imputations for different chromosomes

**Description**

Join multiple genotype imputation objects, as produced by `sim_geno()`, for the same individuals but different chromosomes.

**Usage**

```r
## S3 method for class 'sim_geno'
cbind(...)  
```

**Arguments**

- `...` Genotype imputation objects as produced by `sim_geno()`. Must have the same set of individuals.

**Value**

An object of class "sim_geno", like the input; see `sim_geno()`.

**See Also**

`rbind.sim_geno()`, `sim_geno()`

**Examples**

```r
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
map <- insert_pseudomarkers(grav2$gmap, step=1)
drawsA <- sim_geno(grav2[1:5,1:2], map, error_prob=0.002, n_draws=4)
drawsB <- sim_geno(grav2[1:5,3:4], map, error_prob=0.002, n_draws=4)
draws <- cbind(drawsA, drawsB)
```
**cbind.viterbi**  
*Join viterbi results for different chromosomes*

**Description**
Join multiple viterbi objects, as produced by `viterbi()`, for the same set of individuals but different chromosomes.

**Usage**
```r
## S3 method for class 'viterbi'
cbind(...)  
```

**Arguments**
- `...` Imputed genotype objects as produced by `viterbi()`. Must have the same set of individuals.

**Value**
An object of class "viterbi", like the input; see `viterbi()`.

**See Also**
`rbind.viterbi()`, `viterbi()`

**Examples**
```r
ggrav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
gmap <- insert_pseudomarkers(grav2$gmap, step=1)
gA <- viterbi(grav2[1:5,1:2], gmap, error_prob=0.002)
gB <- viterbi(grav2[1:5,3:4], gmap, error_prob=0.002)
g <- cbind(gA, gB)
```

---

**cbind_expand**  
*Combine matrices by columns, expanding and aligning rows*

**Description**
This is like `base::cbind()` but using row names to align the rows and expanding with missing values if there are rows in some matrices but not others.

**Usage**
```r
cbind_expand(...)  
```
Arguments

A set of matrices or data frames

Value

The matrices combined by columns, using row names to align the rows, and expanding with missing values if there are rows in some matrices but not others.

Examples

df1 <- data.frame(x=c(1,2,3,NA,4), y=c(5,8,9,10,11), row.names=c("A", "B", "C", "D", "E"))
df2 <- data.frame(w=c(7,8,0,9,10), z=c(6,NA,NA,9,10), row.names=c("A", "B", "F", "C", "D"))
cbind_expand(df1, df2)

<table>
<thead>
<tr>
<th>CCcolors</th>
<th>Collaborative Cross colors</th>
</tr>
</thead>
</table>

Description

A vector of 8 colors for use with the mouse Collaborative Cross and Diversity Outbreds.

Details

CCorigcolors are the original eight colors for the Collaborative Cross founder strains. CCaltcolors are a slightly modified version, but still not color-blind friendly. CCcolors are derived from the Okabe-Ito color blind friendly palette in Wong (2011) Nature Methods doi:10.1038/nmeth.1618.

Source

https://csbio.unc.edu/CCstatus/index.py?run=AvailableLines.information

Examples

data(CCcolors)
data(CCaltcolors)
data(CCorigcolors)
check_cross2  

**Check a cross2 object**

**Description**

Check the integrity of the data within a cross2 object.

**Usage**

```r
check_cross2(cross2)
```

**Arguments**

- `cross2`  
  An object of class "cross2", as output by `read_cross2()`. For details, see the `R/qtl2` developer guide.

**Details**

Checks whether a cross2 object meets the specifications. Problems are issued as warnings.

**Value**

If everything is correct, returns TRUE; otherwise FALSE, with attributes that give the problems.

**Examples**

```r
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
check_cross2(grav2)
```

chisq_colpairs  

**Chi-square test on all pairs of columns**

**Description**

Perform a chi-square test for independence for all pairs of columns of a matrix.

**Usage**

```r
chisq_colpairs(x)
```

**Arguments**

- `x`  
  A matrix of positive integers. NAs and values <= 0 are treated as missing.
Value

A matrix of size p x p, where p is the number of columns in the input matrix x, containing the chi-square test statistics for independence, applied to pairs of columns of x. The diagonal of the result will be all NAs.

Examples

```r
z <- matrix(sample(1:2, 500, replace=TRUE), ncol=5)
chisq_colpairs(z)
```

chr_lengths

Calculate chromosome lengths

Description

Calculate chromosome lengths for a map object

Usage

`chr_lengths(map, collapse_to_AX = FALSE)`

Arguments

map

A list of vectors, each specifying locations of the markers.

collapse_to_AX

If TRUE, collapse to the total lengths of the autosomes and X chromosome.

Details

We take `diff(range(v))` for each vector, v.

Value

A vector of chromosome lengths. If `collapse_to_AX=TRUE`, the result is a vector of length 2 (autosomal and X chromosome lengths).

See Also

`scan1perm()`
**clean**

Clean an object

**Description**

Clean an object by removing messy values

**Usage**

```r
clean(object, ...)  
```

**Arguments**

- `object` Object to be cleaned
- `...` Other arguments

**Value**

Input object with messy values cleaned up

**See Also**

- `clean.scan1()`, `clean.calc_genoprob()`

**clean_genoprob**

Clean genotype probabilities

**Description**

Clean up genotype probabilities by setting small values to 0 and for a genotype column where the maximum value is rather small, set all values in that column to 0.

**Usage**

```r
clean_genoprob(  
  object,  
  value_threshold = 0.000001,  
  column_threshold = 0.01,  
  ind = NULL,  
  cores = 1,  
  ...  
)  
```

```r  
# S3 method for class 'calc_genoprob'  
clean(  
```
clean_genoprob

function (object, value_threshold = 0.000001, column_threshold = 0.01, ind = NULL, cores = 1, ...
)

Arguments

object Genotype probabilities as calculated by `calc_genoprob()`.
value_threshold Probabilities below this value will be set to 0.
column_threshold For genotype columns where the maximum value is below this threshold, all values will be set to 0. This must be less than \(1/k\) where \(k\) is the number of genotypes.
ind Optional vector of individuals (logical, numeric, or character). If provided, only the genotype probabilities for these individuals will be cleaned, though the full set will be returned.
cores Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`.) Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.
...
Ignored at this point.

Details

In cases where a particular genotype is largely absent, `scan1coef()` and `fit1()` can give unstable estimates of the genotype effects. Cleaning up the genotype probabilities by setting small values to 0 helps to ensure that such effects get set to `NA`.

At each position and for each genotype column, we find the maximum probability across individuals. If that maximum is < `column_threshold`, all values in that genotype column at that position are set to 0.

In addition, any genotype probabilities that are < `value_threshold` (generally < `column_threshold`) are set to 0.

The probabilities are then re-scaled so that the probabilities for each individual at each position sum to 1.

If `ind` is provided, the function is applied only to the designated subset of individuals. This may be useful when only a subset of individuals have been phenotyped, as you may want to zero out genotype columns where that subset of individuals has only negligible probability values.

Value

A cleaned version of the input genotype probabilities object, `object`. 
Examples

iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# calculate genotype probabilities
probs <- calc_genoprob(iron, error_prob=0.002)

# clean the genotype probabilities
# (doesn't really do anything in this case, because there are no small but non-zero values)
probs_clean <- clean(probs)

# clean only the females' genotype probabilities
probs_cleanf <- clean(probs, ind=names(iron$is_female)[iron$is_female])

---

clean_scan1  Clean scan1 output

Description

Clean scan1 output by replacing negative values with NA and remove rows where all values are NA.

Usage

clean_scan1(object, ...)

# S3 method for class 'scan1'
clean(object, ...)

Arguments

object  Output of scan1().
...

Ignored at present

Value

The input object with negative values replaced with NAs and then rows with all NAs removed.

Examples

iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

pr <- calc_genoprob(iron)
out <- scan1(pr, iron$pheno)

out <- clean(out)
**compare_gen**

---

**Description**

Count the number of matching genotypes between all pairs of individuals, to look for unusually closely related individuals.

**Usage**

```r
compare_gen(cross, omit_x = FALSE, proportion = TRUE, quiet = TRUE, cores = 1)
```

**Arguments**

- `cross`: Object of class "cross2". For details, see the R/qtl2 developer guide.
- `omit_x`: If TRUE, only use autosomal genotypes
- `proportion`: If TRUE (the default), the upper triangle of the result contains the proportions of matching genotypes. If FALSE, the upper triangle contains counts of matching genotypes.
- `quiet`: IF FALSE, print progress messages.
- `cores`: Number of CPU cores to use, for parallel calculations. (If 0, use parallel::detectCores().) Alternatively, this can be links to a set of cluster sockets, as produced by parallel::makeCluster().

**Value**

A square matrix; diagonal is number of observed genotypes for each individual. The values in the lower triangle are the numbers of markers where both of a pair were genotyped. The values in the upper triangle are either proportions or counts of matching genotypes for each pair (depending on whether proportion=TRUE or =FALSE). The object is given class "compare_gen".

**Examples**

```r
gav2 <- read_cross2(system.file("extdata", "gav2.zip", package="qtl2"))
cg <- compare_gen(cross="gav2")
summary(cg)
```
**compare_genoprob** \(\textit{Compare two sets of genotype probabilities}\)

**Description**

Compare two sets of genotype probabilities for one individual on a single chromosome.

**Usage**

```r
compare_genoprob(
  probs1,
  probs2,
  cross,
  ind = 1,
  chr = NULL,
  minprob = 0.95,
  minmarkers = 10,
  minwidth = 0,
  annotate = FALSE
)
```

**Arguments**

- `probs1`: Genotype probabilities (as produced by `calc_genoprob()` or allele dosages (as produced by `genoprob_to_alleleprob()`).
- `probs2`: A second set of genotype probabilities, just like `probs1`.
- `cross`: Object of class "cross2". For details, see the `R/qtl2` developer guide.
- `ind`: Individual to plot, either a numeric index or an ID.
- `chr`: Selected chromosome; a single character string.
- `minprob`: Minimum probability for inferring genotypes (passed to `maxmarg()`).
- `minmarkers`: Minimum number of markers in results.
- `minwidth`: Minimum width in results.
- `annotate`: If TRUE, add some annotations to the `geno1` and `geno2` columns to indicate, where they differ, which one matches what appears to be the best genotype. (* = matches the best genotype; - = lower match).

**Details**

The function does the following:

- Reduce the probabilities to a set of common locations that also appear in `cross`.
- Use `maxmarg()` to infer the genotype at every position using each set of probabilities.
- Identify intervals where the two inferred genotypes are constant.
- Within each segment, compare the observed SNP genotypes to the founders’ genotypes.
**compare_maps**

**Value**

A data frame with each row corresponding to an interval over which `probs1` and `probs2` each have a fixed inferred genotype. Columns include the two inferred genotypes, the start and end points and width of the interval, and when founder genotypes are in cross, the proportions of SNPs where the individual matches each possible genotypes.

**See Also**

`plot_genoprobcomp()`

**Examples**

```r
ingo <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
ingo <- iron[1,"2"]  # subset to first individual on chr 2
map <- insert_pseudomarkers(iron$map, step=1)

# in presence of a genotyping error, how much does error_prob matter?
ingo$geno[1][1,3] <- 3
pr_e <- calc_genoprob(iron, map, error_prob=0.002)
pr_ne <- calc_genoprob(iron, map, error_prob=1e-15)

compare_genoprob(pr_e, pr_ne, iron, minmarkers=1, minprob=0.5)
```
Value
A data frame containing

- marker - marker name
- chr_map1 - chromosome ID on map1
- pos_map1 - position on map1
- chr_map2 - chromosome ID on map2
- pos_map2 - position on map2

Examples

```r
# load some data
iron <- read.cross2( system.file("extdata", "iron.zip", package="qtl2") )
gmap <- iron$gmap
pmap <- iron$pmap

# omit a marker from each map
gmap[[7]] <- gmap[[7]][-3]
pmap[[8]] <- pmap[[8]][-7]

# swap order of a couple of markers on the physical map
names(pmap[[9]])[3:4] <- names(pmap[[9]])[4:3]

# move a marker to a different chromosome
pmap[[10]] <- c(pmap[[10]], pmap[[11]][2])[c(1,3,2)]
pmap[[1]] <- pmap[[1]][-2]

# compare these messed-up maps
compare_maps(gmap, pmap)
```

---

**convert2cross2**  
*Convert R/qtl cross object to new format*

**Description**
Convert a cross object from the R/qtl format to the R/qtl2 format

**Usage**

```r
convert2cross2(cross)
```

**Arguments**

cross  
An object of class "cross"; see `qtl::read.cross()` for details.

**Value**
Object of class "cross2". For details, see the *R/qtl2 developer guide*.
count_xo

See Also

read_cross2()

Examples

```r
library(qtl)
data(hyper)
hyper2 <- convert2cross2(hyper)
```

count_xo Count numbers of crossovers

Description

Estimate the numbers of crossovers in each individual on each chromosome.

Usage

```r
count_xo(geno, quiet = TRUE, cores = 1)
```

Arguments

- **geno**: List of matrices of genotypes (output of `maxmarg()` or `viterbi()`) or a list of 3d-arrays of genotypes (output of `sim_geno()`).
- **quiet**: If FALSE, print progress messages.
- **cores**: Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`.) Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.

Value

A matrix of crossover counts, individuals x chromosomes, or (if the input was the output of `sim_geno()`) a 3d-array of crossover counts, individuals x chromosomes x imputations.

See Also

locate_xo()

Examples

```r
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
map <- insert_pseudomarkers(iron$gmap, step=1)
pr <- calc_genoprob(iron, map, error_prob=0.002, map_function="c-f")
g <- maxmarg(pr)
n_xo <- count_xo(g)
# imputations
imp <- sim_geno(iron, map, error_prob=0.002, map_function="c-f", n_draws=32)
```
n_xo_imp <- count_xo(imp)
# sums across chromosomes
tot_xo_imp <- apply(n_xo_imp, c(1,3), sum)
# mean and SD across imputations
summary_xo <- cbind(mean=rowMeans(tot_xo_imp),
                   sd=apply(tot_xo_imp, 1, sd))

create_gene_query_func

Create a function to query genes

Description
Create a function that will connect to a SQLite database of gene information and return a data frame with gene information for a selected region.

Usage
create_gene_query_func(
  dbfile = NULL,
  db = NULL,
  table_name = "genes",
  chr_field = "chr",
  start_field = "start",
  stop_field = "stop",
  name_field = "Name",
  strand_field = "strand",
  filter = NULL
)

Arguments
dbfile Name of database file
db Optional database connection (provide one of file and db).
table_name Name of table in the database
chr_field Name of chromosome field
start_field Name of field with start position (in basepairs)
stop_field Name of field with stop position (in basepairs)
name_field Name of field with gene name
strand_field Name of field with strand (+/-)
filter Additional SQL filter (as a character string).
create_snpinfo

Details

Note that this function assumes that the database has start and stop fields that are in basepairs, but the selection uses positions in Mbp, and the output data frame should have start and stop columns in Mbp.

Also note that a SQLite database of MGI mouse genes is available at figshare: doi:10.6084/m9.figshare.5286019.v7

Value

Function with three arguments, chr, start, and end, which returns a data frame with the genes overlapping that region, with start and end being in Mbp. The output should contain at least the columns Name, chr, start, and stop, the latter two being positions in Mbp.

Examples

# create query function by connecting to file
dbfile <- system.file("extdata", "mouse_genes_small.sqlite", package="qtl2")
query_genes <- create_gene_query_func(dbfile, filter="(source=='MGI')")
# query_genes will connect and disconnect each time
genes <- query_genes("2", 97.0, 98.0)

# connect and disconnect separately
library(RSQLite)
db <- dbConnect(SQLite(), dbfile)
query_genes <- create_gene_query_func(db=db, filter="(source=='MGI')")
genes <- query_genes("2", 97.0, 98.0)
dbDisconnect(db)

create_snpinfo

Create snp information table for a cross

Description

Create a table of snp information from a cross, for use with scan1snps().

Usage

create_snpinfo(cross)

Arguments

cross Object of class "cross2". For details, see the R/qtl2 developer guide.
create_variant_query_func

Create a function to query variants

Description

Create a function that will connect to a SQLite database of founder variant information and return a data frame with variants for a selected region.

Value

A data frame of SNP information with the following columns:

- **chr** - Character string or factor with chromosome
- **pos** - Position (in same units as in the "map" attribute in genoprobs.
- **snp** - Character string with SNP identifier (if missing, the rownames are used).
- **sdp** - Strain distribution pattern: an integer, between 1 and \(2^n - 2\) where \(n\) is the number of strains, whose binary encoding indicates the founder genotypes SNPs with missing founder genotypes are omitted.

See Also

`index_snps()`, `scan1snps()`, `genoprob_to_snpprob()`

Examples

```r
## Not run:
# load example data and calculate genotype probabilities
file <- paste0("https://raw.githubusercontent.com/rqtl/",
"qtl2data/main/DO_Recla/recla.zip")
recla <- read_cross2(file)
snpinfo <- create.snpinfo(recla)

# calculate genotype probabilities
pr <- calc_genoprob(recla, error_prob=0.002, map_function="c-f")

# index the snp information
snpinfo <- index_snps(recla$pmap, snpinfo)

# sex covariate
sex <- setNames((recla$covar$Sex=="female")*1, rownames(recla$covar))

# perform a SNP scan
out <- scan1snps(pr, recla$pmap, recla$pheno[,"bw"], addcovar=sex, snpinfo=snpinfo)

# plot the LOD scores
plot(out$lod, snpinfo, altcol="green3")

## End(Not run)
```
create_variant_query_func

Usage

create_variant_query_func(
    dbfile = NULL,
    db = NULL,
    table_name = "variants",
    chr_field = "chr",
    pos_field = "pos",
    id_field = "snp_id",
    sdp_field = "sdp",
    filter = NULL
)

Arguments

dbfile Name of database file
db Optional database connection (provide one of file and db).
table_name Name of table in the database
chr_field Name of chromosome field
pos_field Name of position field
id_field Name of SNP/variant ID field
sdp_field Name of strain distribution pattern (SDP) field
filter Additional SQL filter (as a character string)

Details

Note that this function assumes that the database has a pos field that is in basepairs, but the selection uses start and end positions in Mbp, and the output data frame should have pos in Mbp.

Also note that a SQLite database of variants in the founder strains of the mouse Collaborative Cross is available at figshare: doi:10.6084/m9.figshare.5280229.v3

Value

Function with three arguments, chr, start, and end, which returns a data frame with the variants in that region, with start and end being in Mbp. The output should contain at least the columns chr and pos, the latter being position in Mbp.

Examples

# create query function by connecting to file
dbfile <- system.file("extdata", "cc_variants_small.sqlite", package="qtl2")
query_variants <- create_variant_query_func(dbfile)
# query_variants will connect and disconnect each time
variants <- query_variants("2", 97.0, 98.0)

# create query function to just grab SNPs
query_snps <- create_variant_query_func(dbfile, filter="type=='snp'")
# query_variants will connect and disconnect each time
snps <- query_snps("2", 97.0, 98.0)

# connect and disconnect separately
library(RSQLite)
db <- dbConnect(SQLite(), dbfile)
query_variants <- create_variant_query_func(db=db)
variants <- query_variants("2", 97.0, 98.0)
dbDisconnect(db)

decomp_kinship

Calculate eigen decomposition of kinship matrix

Description

Calculate the eigen decomposition of a kinship matrix, or of a list of such matrices.

Usage

decomp_kinship(kinship, cores = 1)

Arguments

kinship A square matrix, or a list of square matrices.
cores Number of CPU cores to use, for parallel calculations. (If 0, use parallel::detectCores().) Alternatively, this can be links to a set of cluster sockets, as produced by parallel::makeCluster().

Details

The result contains an attribute "eigen_decomp".

Value

The eigen values and the transposed eigen vectors, as a list containing a vector values and a matrix vectors.

Examples

iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

map <- insert_pseudomarkers(iron$gmap, step=1)
probs <- calc_genoprob(iron, map, error_prob=0.002)
K <- calc_kinship(probs)

Ke <- decomp_kinship(K)
drop_markers  

Drop markers from a cross2 object

Description

Drop a vector of markers from a cross2 object.

Usage

drop_markers(cross, markers)

Arguments

cross Object of class "cross2". For details, see the R/qtl2 developer guide.
markers A vector of marker names.

Value

The input cross with the specified markers removed.

See Also

pull_markers(), drop_nullmarkers()

Examples

grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
markers2drop <- c("BH.342C/347L-Col", "GH.94L", "EG.357C/359L-Col", "CD.245L", "ANL2")
grav2_rev <- drop_markers(grav2, markers2drop)

drop_nullmarkers  

Drop markers with no genotype data

Description

Drop markers with no genotype data (or no informative genotypes)

Usage

drop_nullmarkers(cross, quiet = FALSE)

Arguments

cross Object of class "cross2". For details, see the R/qtl2 developer guide.
quiet If FALSE, print information about how many markers were dropped.
Details

We omit any markers that have completely missing data, or if founder genotypes are present (e.g., for Diversity Outbreds), the founder genotypes are missing or are all the same.

Value

The input cross with the uninformative markers removed.

See Also

drop_markers(), pull_markers()

Examples

```r
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
# make a couple of markers missing
grav2$geno[[2]][,c(3,25)] <- 0
grav2_rev <- drop_nullmarkers(grav2)
```

---

**est_herit**

*Estimate heritability with a linear mixed model*

Description

Estimate the heritability of a set of traits via a linear mixed model, with possible allowance for covariates.

Usage

```r
est_herit(
  pheno,
  kinship,
  addcovar = NULL,
  weights = NULL,
  reml = TRUE,
  cores = 1,
  ...
)
```

Arguments

- `pheno` A numeric matrix of phenotypes, individuals x phenotypes.
- `kinship` A kinship matrix.
- `addcovar` An optional numeric matrix of additive covariates.
- `weights` An optional numeric vector of positive weights for the individuals. As with the other inputs, it must have names for individual identifiers.
If true, use REML; otherwise, use maximum likelihood.  

Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`.) Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.

... Additional control parameters (see details).

**Details**

We fit the model $y = X\beta + \epsilon$ where $\epsilon$ is multivariate normal with mean 0 and covariance matrix $\sigma^2[h^2(2K) + I]$ where $K$ is the kinship matrix and $I$ is the identity matrix.

For each of the inputs, the row names are used as individual identifiers, to align individuals.

If `reml=TRUE`, restricted maximum likelihood (REML) is used to estimate the heritability, separately for each phenotype.

Additional control parameters include `tol` for the tolerance for convergence, `quiet` for controlling whether messages will be display, `max_batch` for the maximum number of phenotypes in a batch, and `check_boundary` for whether the 0 and 1 boundary values for the estimated heritability will be checked explicitly.

**Value**

A vector of estimated heritabilities, corresponding to the columns in `pheno`. The result has attributes "sample_size", "log10lik" and "resid_sd".

**Examples**

```r
# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# kinship matrix
kinship <- calc_kinship(probs)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)

# perform genome scan
hsq <- est_herit(pheno, kinship, covar)
```
Description

Uses a hidden Markov model to re-estimate the genetic map for an experimental cross, with possible allowance for genotyping errors.

Usage

```r
est_map(
  cross,
  error_prob = 0.0001,
  map_function = c("haldane", "kosambi", "c-f", "morgan"),
  lowmem = FALSE,
  maxit = 10000,
  tol = 0.000001,
  quiet = TRUE,
  save_rf = FALSE,
  cores = 1
)
```

Arguments

- `cross` Object of class "cross2". For details, see the R/qtl2 developer guide.
- `error_prob` Assumed genotyping error probability
- `map_function` Character string indicating the map function to use to convert genetic distances to recombination fractions.
- `lowmem` If FALSE, precalculate initial and emission probabilities, and at each iteration calculate the transition matrices for a chromosome; potentially a lot faster but using more memory. Needs to be tailored somewhat to cross type. For example, multi-way RIL may need to reorder the transition matrix according to cross order, and AIL and DO need separate transition matrices for each generation.
- `maxit` Maximum number of iterations in EM algorithm.
- `tol` Tolerance for determining convergence
- `quiet` If FALSE, print progress messages.
- `save_rf` If TRUE, save the estimated recombination fractions as an attribute ("rf") of the result.
- `cores` Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`.) Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.

Details

The map is estimated assuming no crossover interference, but a map function (by default, Haldane’s) is used to derive the genetic distances.
find_ibd_segments

Value
A list of numeric vectors, with the estimated marker locations (in cM). The location of the initial marker on each chromosome is kept the same as in the input cross.

Examples
```
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
gmap <- est_map(grav2, error_prob=0.002)
```

Description
Find IBD segments (regions with a lot of shared SNP genotypes) for a set of strains

Usage
```
find_ibd_segments(geno, map, min_lod = 15, error_prob = 0.001, cores = 1)
```

Arguments
- **geno**: List of matrices of founder genotypes. The matrices correspond to the genotypes on chromosomes and are arrayed as founders x markers.
- **map**: List of vectors of marker positions
- **min_lod**: Threshold for minimum LOD score for a segment
- **error_prob**: Genotyping error/mutation probability
- **cores**: Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`. Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.

Details
For each strain pair on each chromosome, we consider all marker intervals and calculate a LOD score comparing the two hypotheses: that the strains are IBD in the interval, vs. that they are not. We assume that the two strains are homozygous at all markers, and use the model from Broman and Weber (1999), which assumes linkage equilibrium between markers and uses a simple model for genotype frequencies in the presence of genotyping errors or mutations.

Note that inference of IBD segments is heavily dependent on how SNPs were chosen to be genotyped. (For example, were the SNPs ascertained based on their polymorphism between a particular strain pair?)
Value

A data frame whose rows are IBD segments and whose columns are:

- Strain 1
- Strain 2
- Chromosome
- Left marker
- Right marker
- Left position
- Right position
- Left marker index
- Right marker index
- Interval length
- Number of markers
- Number of mismatches
- LOD score

References


Examples

```r
## Not run:

# grab founder genotypes and physical map
fg <- recla$founder_geno
pmap <- recla$pmap

# find shared segments
(segs <- find_ibd_segments(fg, pmap, min_lod=10, error_prob=0.0001))

## End(Not run)
```
find_index_snp

Find name of indexed SNP

Description
For a particular SNP, find the name of the corresponding indexed SNP.

Usage
find_index_snp(snpinfo, snp)

Arguments
snpinfo Data frame with SNP information with the following columns:
• chr - Character string or factor with chromosome
• index - Numeric index of equivalent, indexed SNP, as produced by `index_snps()`.
• snp - Character string with SNP identifier (if missing, the rownames are used).

snp Name of snp to look for (can be a vector).

Value
A vector of SNP IDs (the corresponding indexed SNPs), with NA if a SNP is not found.

See Also
`find_marker()`

Examples
```r
## Not run:
# load example data and calculate genotype probabilities
file <- paste0("https://raw.githubusercontent.com/rqtl/",
"qtl2data/main/DO_Recla/recla.zip")
recla <- read_cross2(file)
# founder genotypes for a set of SNPs
snpgeno <- rbind(m1=c(3,1,1,3,1,1,1,1),
                 m2=c(3,1,1,3,1,1,1,1),
                 m3=c(1,1,1,1,3,3,3,3),
                 m4=c(1,3,1,3,1,3,1,3))
sdp <- calc_sdp(snpgeno)
# update snp info by adding the SNP index column
```
find_map_gaps

Find gaps in a genetic map

Description
Find gaps between markers in a genetic map

Usage
find_map_gaps(map, min_gap = 50)

Arguments
map Genetic map as a list of vectors (each vector is a chromosome and contains the marker positions).
min_gap Minimum gap length to return.

Value
Data frame with 6 columns: chromosome, marker to left of gap, numeric index of marker to left, marker to right of gap, numeric index of marker to right, and the length of the gap.

See Also
reduce_map_gaps()

Examples
iron <- read.cross2(system.file("extdata", "iron.zip", package="qtl2"))
find_map_gaps(iron$gmap, 40)
find_marker

Find markers by chromosome position

Description

Find markers closest to specified set of positions, or within a specified interval.

Usage

find_marker(map, chr, pos = NULL, interval = NULL)

Arguments

map
A map object: a list (corresponding to chromosomes) of vectors of marker positions. Can also be a snpinfo object (data frame with columns chr and pos; marker names taken from column snp or if that doesn’t exist from the row names)

chr
A vector of chromosomes

pos
A vector of positions

interval
A pair of positions (provide either pos or interval but not both)

Details

If pos is provided, interval should not be, and vice versa.
If pos is provided, then chr and pos should either be the same length, or one of them should have length 1 (to be expanded to the length of the other).
If interval is provided, then chr should have length 1.

Value

A vector of marker names, either closest to the positions specified by pos, or within the interval defined by interval.

See Also

find_markerpos(), find_index_snp(), pull_genoprobpos(), pull_genoprobint()

Examples

iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# find markers by their genetic map positions
find_marker(iron$gmap, c(8, 11), c(37.7, 56.9))

# find markers by their physical map positions (two markers on chr 7)
find_marker(iron$pmap, 7, c(44.2, 108.9))
find_markerpos

# find markers in an interval
find_marker(iron$pmap, 16, interval=c(35, 80))

find_markerpos  Find positions of markers

Description

Find positions of markers within a cross object

Usage

find_markerpos(cross, markers, na.rm = TRUE)

Arguments

cross Object of class "cross2". For details, see the R/qtl2 developer guide. Can also be a map (as a list of vectors of marker positions).
markers A vector of marker names.
na.rm If TRUE, don’t include not-found markers in the results (but issue a warning if some markers weren’t found). If FALSE, include those markers with NA for chr and position.

Value

A data frame with chromosome and genetic and physical positions (in columns "gmap" and "pmap"), with markers as row names. If the input cross is not a cross2 object but rather a map, the output contains chr and pos.

See Also

find_marker()

Examples

iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# find markers
find_markerpos(iron, c("D8Mit294", "D11Mit101"))
find_peaks

Find peaks in a set of LOD curves (output from scan1())

Usage

find_peaks(
  scan1_output,
  map,
  threshold = 3,
  peakdrop = Inf,
  drop = NULL,
  prob = NULL,
  thresholdX = NULL,
  peakdropX = NULL,
  dropX = NULL,
  probX = NULL,
  expand2markers = TRUE,
  sort_by = c("column", "pos", "lod"),
  cores = 1
)

Arguments

scan1_output An object of class "scan1" as returned by scan1().
map A list of vectors of marker positions, as produced by insert_pseudomarkers(). Can also be an indexed SNP info table, as from index_snps() or scan1snps().
threshold Minimum LOD score for a peak (can be a vector with separate thresholds for each lod score column in scan1_output).
peakdrop Amount that the LOD score must drop between peaks, if multiple peaks are to be defined on a chromosome. (Can be a vector with separate values for each lod score column in scan1_output.)
don If provided, LOD support intervals are included in the results, and this indicates the amount to drop in the support interval. (Can be a vector with separate values for each lod score column in scan1_output.) Must be ≤ peakdrop.
prob If provided, Bayes credible intervals are included in the results, and this indicates the nominal coverage. (Can be a vector with separate values for each lod score column in scan1_output.) Provide just one of drop and prob.
thresholdX Separate threshold for the X chromosome; if unspecified, the same threshold is used for both autosomes and the X chromosome. (Like threshold, this can be a vector with separate thresholds for each lod score column.)
peakdropX  Like peakdrop, but for the X chromosome; if unspecified, the same value is used for both autosomes and the X chromosome. (Can be a vector with separate values for each lod score column in scan1_output.)

dropX  Amount to drop for LOD support intervals on the X chromosome. Ignored if drop is not provided. (Can be a vector with separate values for each lod score column in scan1_output.)

probX  Nominal coverage for Bayes intervals on the X chromosome. Ignored if prob is not provided. (Can be a vector with separate values for each lod score column in scan1_output.)

expand2markers  If TRUE (and if drop or prob is provided, so that QTL intervals are calculated), QTL intervals are expanded so that their endpoints are at genetic markers.

sort_by  Indicates whether to sort the rows by lod column, genomic position, or LOD score.

tores  Number of CPU cores to use, for parallel calculations. (If 0, use parallel::detectCores(). Alternatively, this can be links to a set of cluster sockets, as produced by parallel::makeCluster().

Details
For each lod score column on each chromosome, we return a set of peaks defined as local maxima that exceed the specified threshold, with the requirement that the LOD score must have dropped by at least peakdrop below the lowest of any two adjacent peaks.

At a given peak, if there are ties, with multiple positions jointly achieving the maximum LOD score, we take the average of these positions as the location of the peak.

Value
A data frame with each row being a single peak on a single chromosome for a single LOD score column, and with columns

- lodindex - lod column index
- lodcolumn - lod column name
- chr - chromosome ID
- pos - peak position
- lod - lod score at peak

If drop or prob is provided, the results will include two additional columns: ci_lo and ci_hi, with the endpoints of the LOD support intervals or Bayes credible wintervals.

See Also
scan1(), lod_int(), bayes_int()
Examples

# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)

# perform genome scan
out <- scan1(probs, pheno, addcovar=covar, Xcovar=Xcovar)

# find just the highest peak on each chromosome
find_peaks(out, map, threshold=3)

# possibly multiple peaks per chromosome
find_peaks(out, map, threshold=3, peakdrop=1)

# possibly multiple peaks, also getting 1-LOD support intervals
find_peaks(out, map, threshold=3, peakdrop=1, drop=1)

# possibly multiple peaks, also getting 90% Bayes intervals
find_peaks(out, map, threshold=3, peakdrop=1, prob=0.9)

fit1

Fit single-QTL model at a single position

Description

Fit a single-QTL model at a single putative QTL position and get detailed results about estimated coefficients and individuals contributions to the LOD score.

Usage

fit1(
  genoprobs,
  pheno,
  kinship = NULL,
  addcovar = NULL,
  nullcovar = NULL,
  intcovar = NULL,
)
weights = NULL,
contrasts = NULL,
model = c("normal", "binary"),
zerosum = TRUE,
se = TRUE,
hsq = NULL,
reml = TRUE,
blup = FALSE,
...)

Arguments

genoprobs A matrix of genotype probabilities, individuals x genotypes. If NULL, we create a single intercept column, matching the individual IDs in pheno.

pheno A numeric vector of phenotype values (just one phenotype, not a matrix of them)

kinship Optional kinship matrix.

addcovar An optional numeric matrix of additive covariates.

tnullcovar An optional numeric matrix of additional additive covariates that are used under the null hypothesis (of no QTL) but not under the alternative (with a QTL). This is needed for the X chromosome, where we might need sex as a additive covariate under the null hypothesis, but we wouldn’t want to include it under the alternative as it would be collinear with the QTL effects.

intcovar An optional numeric matrix of interactive covariates.

weights An optional numeric vector of positive weights for the individuals. As with the other inputs, it must have names for individual identifiers.

contrasts An optional numeric matrix of genotype contrasts, size genotypes x genotypes. For an intercross, you might use cbind(mu=c(1,1,1), a=c(-1, 0, 1), d=c(0, 1, 0)) to get mean, additive effect, and dominance effect. The default is the identity matrix.

model Indicates whether to use a normal model (least squares) or binary model (logistic regression) for the phenotype. If model="binary", the phenotypes must have values in [0,1].

zerosum If TRUE, force the genotype or allele coefficients sum to 0 by subtracting their mean and add another column with the mean. Ignored if contrasts is provided.

se If TRUE, calculate the standard errors.

hsq (Optional) residual heritability; used only if kinship provided.

reml If kinship provided: if reml=TRUE, use REML; otherwise maximum likelihood.

blup If TRUE, fit a model with QTL effects being random, as in scan1blup().

... Additional control parameters; see Details;

Details

For each of the inputs, the row names are used as individual identifiers, to align individuals.
If kinship is absent, Haley-Knott regression is performed. If kinship is provided, a linear mixed
model is used, with a polygenic effect estimated under the null hypothesis of no (major) QTL, and
then taken as fixed as known in the genome scan.

If contrasts is provided, the genotype probability matrix, $P$, is post-multiplied by the contrasts
matrix, $A$, prior to fitting the model. So we use $P \cdot A$ as the $X$ matrix in the model. One might view
the rows of $A^{-1}$ as the set of contrasts, as the estimated effects are the estimated genotype effects
pre-multiplied by $A^{-1}$.

The ... argument can contain several additional control parameters; suspended for simplicity (or
confusion, depending on your point of view). tol is used as a tolerance value for linear regression
by QR decomposition (in determining whether columns are linearly dependent on others and should
be omitted); default 1e-12. maxit is the maximum number of iterations for convergence of the
iterative algorithm used when model=binary. bintol is used as a tolerance for convergence for
the iterative algorithm used when model=binary. eta_max is the maximum value for the "linear
predictor" in the case model="binary" (a bit of a technicality to avoid fitted values exactly at 0 or
1).

Value

A list containing

- coef - Vector of estimated coefficients.
- SE - Vector of estimated standard errors (included if se=TRUE).
- lod - The overall lod score.
- ind_lod - Vector of individual contributions to the LOD score (not provided if kinship is
  used).
- fitted - Fitted values.
- resid - Residuals. If blup==TRUE, only coef and SE are included at present.

References

Haley CS, Knott SA (1992) A simple regression method for mapping quantitative trait loci in line


See Also

pull_genoprobpos(), find_marker()

Examples

# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=5)
# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno[,1]
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)

# scan chromosome 7 to find peak
out <- scan1(probs[,"7"], pheno, addcovar=covar)

# find peak position
max_pos <- max(out, map)

# genoprobs at max position
pr_max <- pull_genoprobpos(probs, map, max_pos$chr, max_pos$pos)

# fit QTL model just at that position
out_fit1 <- fit1(pr_max, pheno, addcovar=covar)

---

**genoprob_to_alleleprob**

*Convert genotype probabilities to allele probabilities*

**Description**

Reduce genotype probabilities (as calculated by `calc_genoprob()`) to allele probabilities.

**Usage**

`genoprob_to_alleleprob(probs, quiet = TRUE, cores = 1)`

**Arguments**

- **probs**: Genotype probabilities, as calculated from `calc_genoprob()`.
- **quiet**
  - IF `FALSE`, print progress messages.
- **cores**
  - Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`. Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.

**Value**

An object of class "calc_genoprob", like the input `probs`, but with probabilities collapsed to alleles rather than genotypes. See `calc_genoprob()`.
**genoprob_to_snpprob**

Convert genotype probabilities to SNP probabilities

**Description**

For multi-parent populations, convert founder genotypes at a set of SNPs to convert founder-based genotype probabilities to SNP genotype probabilities.

**Usage**

```r
genoprob_to_snpprob(genoprobs, snpinfo)
```

**Arguments**

- `genoprobs`: Genotype probabilities as calculated by `calc_genoprob()`.
- `snpinfo`: Data frame with SNP information with the following columns (the last three are generally derived with `index_snps()`):
  - `chr`: Character string or factor with chromosome.
  - `pos`: Position (in same units as in the “map” attribute in genoprobs).
  - `sdp`: Strain distribution pattern: an integer, between 1 and $2^n - 2$ where $n$ is the number of strains, whose binary encoding indicates the founder genotypes.
  - `snp`: Character string with SNP identifier (if missing, the rownames are used).
  - `index`: Indices that indicate equivalent groups of SNPs, calculated by `index_snps()`.
  - `intervals`: Indexes that indicate which marker intervals the SNPs reside.
  - `on_map`: Indicate whether SNP coincides with a marker in the genoprobs.

Alternatively, `snpinfo` can be a object of class "cross2", as output by `read_cross2()`, containing the data for a multi-parent population with founder genotypes, in which case the SNP information for all markers with complete founder genotype data is calculated and then used. But, in this case, the genotype probabilities must be at the markers in the cross.

**Details**

We first split the SNPs by chromosome and use `snpinfo$index` to subset to non-equivalent SNPs. `snpinfo$interval` indicates the intervals in the genotype probabilities that contain each. For SNPs contained within an interval, we use the average of the probabilities for the two endpoints. We then collapse the probabilities according to the strain distribution pattern.
Value

An object of class "calc_genoprob", like the input genoprobs, but with imputed genotype probabilities at the selected SNPs indicated in snpinfo$index. See calc_genoprob().

If the input genoprobs is for allele probabilities, the probs output has just two probability columns (for the two SNP alleles). If the input has a full set of $n(n+1)/2$ probabilities for $n$ strains, the probs output has 3 probabilities (for the three SNP genotypes). If the input has full genotype probabilities for the X chromosome ($n(n+1)/2$ genotypes for the females followed by $n$ hemizygous genotypes for the males), the output has 5 probabilities: the 3 female SNP genotypes followed by the two male hemizygous SNP genotypes.

See Also

index_snps(), calc_genoprob(), scan1snps()

Examples

## Not run:
# load example data and calculate genotype probabilities
file <- paste0("https://raw.githubusercontent.com/rqtl/",
"qtldata/main/D0_Recla/recla.zip")
recla <- read_cross2(file)
recla <- recla[c(1:2,53:54), c("19","X")]
# subset to 4 mice and 2 chromosomes
probs <- calc_genoprob(recla, error_prob=0.002)

# founder genotypes for a set of SNPs
snpgeno <- rbind(m1=c(3,1,1,1,1,1,1,1),
                 m2=c(1,3,1,1,1,1,1,1),
                 m3=c(1,1,1,1,3,3,3,3),
                 m4=c(1,3,1,1,3,1,1,3))
sdp <- calc_sdp(snpgeno)
snpinfo <- data.frame(chr=c("19", "19", "X", "X"),
pos=c(40.36, 40.53, 110.91, 111.21),
sdp=sdp,
snp=c("m1", "m2", "m3", "m4"), stringsAsFactors=FALSE)

# identify groups of equivalent SNPs
snpinfo <- index_snps(recla$pmap, snpinfo)

# collapse to SNP genotype probabilities
snpprobs <- genoprob_to_snpprob(probs, snpinfo)

# could also first convert to allele probs
aprobs <- genoprob_to_alleleprob(probs)
snpaprobs <- genoprob_to_snpprob(aprobs, snpinfo)

## End(Not run)
get_common_ids

Get common set of IDs from objects

Description

For a set objects with IDs as row names (or, for a vector, just names), find the IDs that are present in all of the objects.

Usage

get_common_ids(..., complete.cases = FALSE)

Arguments

...  A set of objects: vectors, lists, matrices, data frames, and/or arrays. If one is a character vector with no names attribute, it’s taken to be a set of IDs, itself.

complete.cases  If TRUE, look at matrices and non-character vectors and keep only individuals with no missing values.

Details

This is used (mostly internally) to align phenotypes, genotype probabilities, and covariates in preparation for a genome scan. The complete.cases argument is used to omit individuals with any missing covariate values.

Value

A vector of character strings for the individuals that are in common.

Examples

```r
x <- matrix(0, nrow=10, ncol=5); rownames(x) <- LETTERS[1:10]
y <- matrix(0, nrow=5, ncol=5); rownames(y) <- LETTERS[(1:5)+7]
z <- LETTERS[5:15]
get_common_ids(x, y, z)

x[8,1] <- NA
get_common_ids(x, y, z)
get_common_ids(x, y, z, complete.cases=TRUE)
```
guess_phase

Guess phase of imputed genotypes

Description

Turn imputed genotypes into phased genotypes along chromosomes by attempting to pick the phase that leads to the fewest recombination events.

Usage

guess_phase(cross, geno, deterministic = FALSE, cores = 1)

guess_phase

Guess phase of imputed genotypes

Description

Turn imputed genotypes into phased genotypes along chromosomes by attempting to pick the phase that leads to the fewest recombination events.

Usage

guess_phase(cross, geno, deterministic = FALSE, cores = 1)

guess_phase

Guess phase of imputed genotypes

Description

Turn imputed genotypes into phased genotypes along chromosomes by attempting to pick the phase that leads to the fewest recombination events.

Usage

guess_phase(cross, geno, deterministic = FALSE, cores = 1)
Arguments

cross Object of class "cross2". For details, see the R/qtl2 developer guide.

geno Imputed genotypes, as a list of matrices, as from maxmarg().

deterministic If TRUE, preferentially put smaller allele first when there's uncertainty. If FALSE, the order of alleles is random in such cases.

cores Number of CPU cores to use, for parallel calculations. (If 0, use parallel::detectCores(). Alternatively, this can be links to a set of cluster sockets, as produced by parallel::makeCluster().

Details

We randomly assign the pair of alleles at the first locus to two haplotypes, and then work left to right, assigning alleles to haplotypes one locus at a time seeking the fewest recombination events. The results are subject to arbitrary and random choices. For example, to the right of a homozygous region, either orientation is equally reasonable.

Value

If input cross is phase-known (e.g., recombinant inbred lines), the output will be the input geno. Otherwise, the output will be a list of three-dimensional arrays of imputed genotypes, individual x position x haplotype (1/2).

See Also

maxmarg()

Examples

iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
gmap <- insert_pseudomarkers(iron$gmap, step=1)
probs <- calc_genoprob(iron, gmap, error_prob=0.002)
imp_geno <- maxmarg(probs)
ph_geno <- guess_phase(iron, imp_geno)

index_snps Create index of equivalent SNPs

Description

For a set of SNPs and a map of marker/pseudomarkers, partition the SNPs into groups that are contained within common intervals and have the same strain distribution pattern, and then create an index to a set of distinct SNPs, one per partition.

Usage

index_snps(map, snpinfo, tol = 0.0000001)
Arguments

map
Physical map of markers and pseudomarkers; generally created from `insert_pseudomarkers()` and used for a set of genotype probabilities (calculated with `calc_genoprob()` that are to be used to interpolate SNP genotype probabilities (with `genoprob_to_snpprob()`).

snpinfo
Data frame with SNP information with the following columns:
- `chr` - Character string or factor with chromosome
- `pos` - Position (in same units as in the "map").
- `sdp` - Strain distribution pattern: an integer, between 1 and $2^n - 2$ where $n$ is the number of strains, whose binary encoding indicates the founder genotypes
- `snp` - Character string with SNP identifier (if missing, the rownames are used).

tol
Tolerance for determining whether a SNP is exactly at a position at which genotype probabilities were already calculated.

Details

We split the SNPs by chromosome and identify the intervals in the map that contain each. For SNPs within `tol` of a position at which the genotype probabilities were calculated, we take the SNP to be at that position. For each marker position or interval, we then partition the SNPs into groups that have distinct strain distribution patterns, and choose a single index SNP for each partition.

Value

A data frame containing the input `snpinfo` with three added columns: "index" (which indicates the groups of equivalent SNPs), "interval" (which indicates the map interval containing the SNP, with values starting at 0), and `on_map` (which indicates that the SNP is within `tol` of a position on the map). The rows get reordered, so that they are ordered by chromosome and position, and the values in the "index" column are by chromosome.

See Also

`genoprob_to_snpprob()`, `scan1snps()`, `find_index_snp()`

Examples

```r
## Not run:
# load example data and calculate genotype probabilities
file <- paste0("https://raw.githubusercontent.com/rqtl/",
                 "qtl2data/main/DO_Recla/recla.zip")
recla <- read_cross2(file)
# founder genotypes for a set of SNPs
snpngen <- rbind(m1=c(3,1,1,3,1,1,1,1),
                 m2=c(1,3,1,3,1,3,1,3),
                 m3=c(1,1,1,1,3,3,3,3),
                 m4=c(1,3,1,3,1,3,1,3))
sdp <- calc_sdp(snpngen)
snpinfo <- data.frame(chr=c("19", "19", "X", "X"),
                      sdp)
```
**insert_pseudomarkers**  
Insert pseudomarkers into a marker map

### Description
Insert pseudomarkers into a map of genetic markers

### Usage

```r
text <- 
  insert_pseudomarkers(
    map, 
    step = 0, 
    off_end = 0, 
    stepwidth = c("fixed", "max"), 
    pseudomarker_map = NULL, 
    tol = 0.01, 
    cores = 1 
  )
```

### Arguments

- **map**: A list of numeric vectors; each vector gives marker positions for a single chromosome.
- **step**: Distance between pseudomarkers and markers; if `step=0` no pseudomarkers are inserted.
- **off_end**: Distance beyond terminal markers in which to insert pseudomarkers.
- **stepwidth**: Indicates whether to use a fixed grid (`stepwidth="fixed"`) or to use the maximal distance between pseudomarkers to ensure that no two adjacent markers/pseudomarkers are more than `step` apart.
- **pseudomarker_map**: A map of pseudomarker locations; if provided the `step`, `off_end`, and `stepwidth` arguments are ignored.
- **tol**: Tolerance for determining whether a pseudomarker would duplicate a marker position.
- **cores**: Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`.) Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.
Details

If `stepwidth="fixed"`, a grid of pseudomarkers is added to the marker map.

If `stepwidth="max"`, a minimal set of pseudomarkers are added, so that the maximum distance between adjacent markers or pseudomarkers is at least step. If two adjacent markers are separated by less than step, no pseudomarkers will be added to the interval. If they are more then step apart, a set of equally-spaced pseudomarkers will be added.

If `pseudomarker_map` is provided, then the step, `off_end`, and `stepwidth` arguments are ignored, and the input `pseudomarker_map` is taken to be the set of pseudomarker positions.

Value

A list like the input map with pseudomarkers inserted. Will also have an attribute "is_x_chr", taken from the input map.

See Also

calc_genoprob(), calc_grid()

Examples

```r
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
gmap_w_pmar <- insert_pseudomarkers(iron$gmap, step=1)
```

interp_genoprob  Intercept genotype probabilities

Description

Linear interpolation of genotype probabilities, mostly to get two sets onto the same map for comparison purposes.

Usage

`interp_genoprob(probs, map, cores = 1)`

Arguments

- `probs`: Genotype probabilities, as calculated from `calc_genoprob()`.
- `map`: List of vectors of map positions.
- `cores`: Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`.) Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.

Details

We reduce probs to the positions present in map and then interpolate the genotype probabilities at additional positions in map by linear interpolation using the two adjacent positions. Off the ends, we just copy over the first or last value unchanged.

In general, it's better to use insert_pseudomarkers() and calc_genoprob() to get genotype probabilities at additional positions along a chromosome. This function is a very crude alternative that was implemented in order to compare genotype probabilities derived by different methods, where we first need to get them onto a common set of positions.

Value

An object of class "calc_genoprob", like the input, but with additional positions present in map. See calc_genoprob().

See Also

calc_genoprob()

Examples

iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
probs <- calc_genoprob(iron, iron$gmap, error_prob=0.002)
# you generally wouldn't want to do this, but this is an illustration
map <- insert_pseudomarkers(iron$gmap, step=1)
probs_map <- interp_genoprob(probs, map)

interp_map

Interpolate between maps

Description

Use interpolate to convert from one map to another

Usage

interp_map(map, oldmap, newmap)

Arguments

map The map to be interpolated; a list of vectors.
oldmap Map with positions in the original scale, as in map.
newmap Map with positions in the new scale.
Value

Object of same form as input map but in the units as in newmap.

Examples

```
# load example data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# positions to interpolate from cM to Mbp
tointerp <- list("7" = c(pos7.1= 5, pos7.2=15, pos7.3=25),
                  "9" = c(pos9.1=20, pos9.2=40))

interp_map(tointerp, iron$gmap, iron$pmap)
```

invert_sdp

Calculate SNP genotype matrix from strain distribution patterns

Description

Calculate the matrix of SNP genotypes from a vector of strain distribution patterns (SDPs).

Usage

```
invert_sdp(sdp, n_strains)
```

Arguments

- `sdp`: Vector of strain distribution patterns (integers between 1 and \( 2^n - 2 \) where \( n \) is the number of strains.
- `n_strains`: Number of strains

Value

Matrix of SNP genotypes, markers x strains, coded as 1 (AA) and 3 (BB). Markers with values other than 1 or 3 are omitted, and monomorphic markers, are omitted.

See Also

`sdp2char()`, `calc_sdp()`

Examples

```
sdp <- c(m1=1, m2=12, m3=240)
invert_sdp(sdp, 8)
```
**locate_xo**

**Locate crossovers**

**Description**

Estimate the locations of crossovers in each individual on each chromosome.

**Usage**

```
locate_xo(geno, map, quiet = TRUE, cores = 1)
```

**Arguments**

- `geno` List of matrices of genotypes (output of `maxmarg()` or `viterbi()`).
- `map` List of vectors with the map positions of the markers.
- `quiet` If FALSE, print progress messages.
- `cores` Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`.
  Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.

**Value**

A list of lists of estimated crossover locations, with crossovers placed at the midpoint of the intervals that contain them.

**See Also**

`count_xo()`

**Examples**

```
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
map <- insert_pseudomarkers(iron$gmap, step=1)
pr <- calc_genoprob(iron, map, error_prob=0.002, map_function="c-f")
g <- maxmarg(pr)
pos <- locate_xo(g, iron$gmap)
```
lod_int  

**Calculate LOD support intervals**

**Description**

Calculate LOD support intervals for a single LOD curve on a single chromosome, with the ability to identify intervals for multiple LOD peaks.

**Usage**

```
lod_int(
    scan1_output,  
    map,  
    chr = NULL,  
    lodcolumn = 1,  
    threshold = 0,  
    peakdrop = Inf,  
    drop = 1.5,  
    expand2markers = TRUE  
)
```

**Arguments**

- **scan1_output**: An object of class "scan1" as returned by `scan1()`.
- **map**: A list of vectors of marker positions, as produced by `insert_pseudomarkers()`.
- **chr**: Chromosome ID to consider (must be a single value).
- **lodcolumn**: LOD score column to consider (must be a single value).
- **threshold**: Minimum LOD score for a peak.
- **peakdrop**: Amount that the LOD score must drop between peaks, if multiple peaks are to be defined on a chromosome.
- **drop**: Amount to drop in the support interval. Must be \( \leq \) peakdrop
- **expand2markers**: If TRUE, QTL intervals are expanded so that their endpoints are at genetic markers.

**Details**

We identify a set of peaks defined as local maxima that exceed the specified `threshold`, with the requirement that the LOD score must have dropped by at least `peakdrop` below the lowest of any two adjacent peaks.

At a given peak, if there are ties, with multiple positions jointly achieving the maximum LOD score, we take the average of these positions as the location of the peak.

The default is to use `threshold=0, peakdrop=Inf, and drop=1.5`. We then return results a single peak, no matter the maximum LOD score, and give a 1.5-LOD support interval.
map_to_grid

Value
A matrix with three columns:

- ci_lo - lower bound of interval
- pos - peak position
- ci_hi - upper bound of interval

Each row corresponds to a different peak.

See Also
bayes_int(), find_peaks(), scan1()

Examples

# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)

# perform genome scan
out <- scan1(probs, pheno, addcovar=covar, Xcovar=Xcovar)

# 1.5-LOD support interval for QTL on chr 7, first phenotype
lod_int(out, map, chr=7, lodcolum=1)

map_to_grid

Subset a map to positions on a grid

Description
Subset a map object to the locations on some grid.

Usage
map_to_grid(map, grid)
**Arguments**

- **map**
  - A list of vectors of marker positions.

- **grid**
  - A list of logical vectors (aligned with `map`), with TRUE indicating the position is on the grid.

**Details**

This is generally for the case of a map created with `insert_pseudomarkers()` with `step>0` and `stepwidth="fixed"`, so that the pseudomarkers form a grid along each chromosome.

**Value**

Same list as input, but subset to just include pseudomarkers along a grid.

**See Also**

- `calc_grid()`, `probs_to_grid()`

**Examples**

```r
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
map_w_pmar <- insert_pseudomarkers(grav2$gmap, step=1)
sapply(map_w_pmar, length)
grid <- calc_grid(grav2$gmap, step=1)
map_sub <- map_to_grid(map_w_pmar, grid)
sapply(map_sub, length)
```

---

**mat2strata**  
*Determine strata based on rows of a matrix*

**Description**

Use the rows of a matrix to define a set of strata for a stratified permutation test.

**Usage**

```r
mat2strata(mat)
```

**Arguments**

- **mat**
  - A covariate matrix, as individuals x covariates.

**Value**

A vector of character strings: for each row of `mat`, we use `base::paste()` with `collapse="|"`.

**See Also**

- `get_x_covar()`, `scan1perm()`
Examples

iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
Xcovar <- get_x_covar(iron)
perm_strata <- mat2strata(Xcovar)

maxlod

Overall maximum LOD score

Description

Find overall maximum LOD score in genome scan results, across all positions and columns.

Usage

maxlod(scan1_output, map = NULL, chr = NULL, lodcolumn = NULL)

Arguments

scan1_output An object of class "scan1" as returned by scan1().
map A list of vectors of marker positions, as produced by insert_pseudomarkers().
chr Optional vector of chromosomes to consider.
lodcolumn An integer or character string indicating the LOD score column, either as a numeric index or column name. If NULL, return maximum for all columns.

Value

A single number: the maximum LOD score across all columns and positions for the selected chromosomes.

Examples

# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)
# perform genome scan
out <- scan1(probs, pheno, addcovar=covar, Xcovar=Xcovar)

# overall maximum
maxlod(out)

# maximum on chromosome 2
maxlod(out, map, "2")

maxmarg

Find genotypes with maximum marginal probabilities

Description
For each individual at each position, find the genotype with the maximum marginal probability.

Usage
maxmarg(
  probs,
  map = NULL,
  minprob = 0.95,
  chr = NULL,
  pos = NULL,
  return_char = FALSE,
  quiet = TRUE,
  cores = 1,
  tol = 0.0000000000001
)

Arguments
probs Genotype probabilities, as calculated from calc_genoprob().
map Map of pseudomarkers in probs. Used only if chr and pos are provided.
minprob Minimum probability for making a call. If maximum probability is less then this value, give NA.
chr If provided (along with pos), consider only the single specified position.
pos If provided (along with chr), consider only the single specified position.
return_char If TRUE, return genotype names as character strings.
quiet IF FALSE, print progress messages.
cores Number of CPU cores to use, for parallel calculations. (If 0, use parallel::detectCores().) Alternatively, this can be links to a set of cluster sockets, as produced by parallel::makeCluster().
tol Tolerance value; genotypes with probability that are within tol of each other are treated as equivalent.
max_compare_geno

Details

If multiple genotypes share the maximum probability, one is chosen at random.

Value

If chr and pos are provided, a vector of genotypes is returned. In this case, map is needed. Otherwise, the result is a object like that returned by viterbi(), A list of two-dimensional arrays of imputed genotypes, individuals x positions. Also includes these attributes:

- crosstype - The cross type of the input cross.
- is_x_chr - Logical vector indicating whether chromosomes are to be treated as the X chromosome or not, from input cross.
- alleles - Vector of allele codes, from input cross.

See Also

sim_geno(), viterbi()

Examples

# load data and calculate genotype probabilities
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
pr <- calc_genoprob(iron, error_prob=0.002)

# full set of imputed genotypes
ginf <- maxmarg(pr)

# imputed genotypes at a fixed position
g <- maxmarg(pr, iron$gmap, chr=8, pos=45.5)

# return genotype names rather than integers
g <- maxmarg(pr, iron$gmap, chr=8, pos=45.5, return_char=TRUE)

max_compare_geno Find pair with most similar genotypes

Description

From results of compare_geno(), show the pair with most similar genotypes.

Usage

max_compare_geno(object, ...)

## S3 method for class 'compare_geno'
max(object, ...)
Arguments

object: A square matrix with genotype comparisons for pairs of individuals, as output by `compare_geno()`.

... Ignored

Value

Data frame with individual pair, proportion matches, number of mismatches, number of matches, and total markers genotyped.

Examples

```r
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
cg <- compare_geno(grav2)
max(cg)
```

---

**max_scan1**  
*Find position with maximum LOD score*

Description

Return data frame with the positions having maximum LOD score for a particular LOD score column

Usage

```r
max_scan1(
  scan1_output,
  map = NULL,
  lodcolumn = 1,
  chr = NULL,
  na.rm = TRUE,
  ...
)
```

## S3 method for class 'scan1'
max(scan1_output, map = NULL, lodcolumn = 1, chr = NULL, na.rm = TRUE, ...)

Arguments

scan1_output: An object of class "scan1" as returned by `scan1()`.

map: A list of vectors of marker positions, as produced by `insert_pseudomarkers()`.
Can also be an indexed SNP info table, as from `index_snps()` or `scan1snps()`.

lodcolumn: An integer or character string indicating the LOD score column, either as a numeric index or column name. If NULL, return maximum for all columns.

chr: Optional vector of chromosomes to consider.

na.rm: Ignored (take to be TRUE)

... Ignored
Value

If map is NULL, the genome-wide maximum LOD score for the selected column is returned. If also lodcolumn is NULL, you get a vector with the maximum LOD for each column.

If map is provided, the return value is a data.frame with three columns: chr, pos, and lod score. But if lodcolumn is NULL, you get the maximum for each lod score column, in the format provided by find_peaks(), so a data.frame with five columns: lodindex, lodcolumn, chr, pos, and lod.

Examples

```r
# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)

# perform genome scan
out <- scan1(probs, pheno, addcovar=covar, Xcovar=Xcovar)

# maximum of first column
max(out, map)

# maximum of spleen column
max(out, map, lodcolumn="spleen")

# maximum of first column on chr 2
max(out, map, chr="2")
```

---

### n_missing

**Count missing genotypes**

**Description**

Number (or proportion) of missing (or non-missing) genotypes by individual or marker

**Usage**

```r
n_missing(  
cross,  
by = c("individual", "marker"),
```
plot_coef

plot QTL effects along chromosome

Description
Plot estimated QTL effects along a chromosomes.

Usage

plot_coef(
  x,
  map,
  columns = NULL,
  col = NULL,
plot_coef

scan1_output = NULL,
add = FALSE,
gap = NULL,
top_panel_prop = 0.65,
legend = NULL,
...
)

plot_coefCC(
  x,
  map,
  columns = 1:8,
  col = qtl2::CCcolors,
  scan1_output = NULL,
  add = FALSE,
  gap = NULL,
  top_panel_prop = 0.65,
  legend = NULL,
  ...
)

## S3 method for class 'scan1coef'
plot(
  x,
  map,
  columns = 1,
  col = NULL,
  scan1_output = NULL,
  add = FALSE,
  gap = NULL,
  top_panel_prop = 0.65,
  legend = NULL,
  ...
)

Arguments

x Estimated QTL effects ("coefficients") as obtained from scan1coef().
map A list of vectors of marker positions, as produced by insert_pseudomarkers().
columns Vector of columns to plot
col Vector of colors, same length as columns. If NULL, some default choices are made.
scan1_output If provided, we make a two-panel plot with coefficients on top and LOD scores below. Should have just one LOD score column; if multiple, only the first is used.
add If TRUE, add to current plot (must have same map and chromosomes).
gap Gap between chromosomes. The default is 1% of the total genome length.
top_panel_prop
If scan1_output provided, this gives the proportion of the plot that is devoted to the top panel.

legend
Location of legend, such as "bottomleft" or "topright" (NULL for no legend)

... Additional graphics parameters.

Details

plot_coefCC() is the same as plot_coef(), but forcing columns=1:8 and using the Collaborative Cross colors, CCcolors.

Value
None.

Hidden graphics parameters

A number of graphics parameters can be passed via .... For example, bgcolor to control the background color, and things like ylab and ylim. These are not included as formal parameters in order to avoid cluttering the function definition.

In the case that scan1_output is provided, col, ylab, and ylim all control the panel with estimated QTL effects, while col_lod, ylab_lod, and ylim_lod control the LOD curve panel.

If legend is indicated so that a legend is shown, legend_lab controls the labels in the legend, and legend_ncol indicates the number of columns in the legend.

See Also

CCcolors, plot_scan1(), plot_snpasso()

Examples

# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno[,1]
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)

# calculate coefficients for chromosome 7
coef <- scan1coef(probs[,7], pheno, addcovar=covar)

# plot QTL effects (note the need to subset the map object, for chromosome 7)
plot(coef, map[7], columns=1:3, col=c("slateblue", "violetred", "green3"))
plot_compare_geno

Plot of compare_geno object.

Description

From results of `compare_geno()`, plot histogram of

Usage

```r
plot_compare_geno(x, rug = TRUE, ...)
## S3 method for class 'compare_geno'
plot(x, rug = TRUE, ...)
```

Arguments

- `x` A square matrix with genotype comparisons for pairs of individuals, as output by `compare_geno()`.
- `rug` If true, use `rug()` to plot tick marks at observed values below the histogram.
- `...` Additional graphics parameters passed to `hist()`

Value

None.

Examples

```r
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
cg <- compare_geno(grav2)
plot(cg)
```

plot_genes

Plot gene locations for a genomic interval

Description

Plot gene locations for a genomic interval, as rectangles with gene symbol (and arrow indicating strand/direction) below.
plot_genes

Usage

plot_genes(
  genes,
  minrow = 4,
  padding = 0.2,
  colors = c("black", "red3", "green4", "blue3", "orange"),
  scale_pos = 1,
  start_field = "start",
  stop_field = "stop",
  strand_field = "strand",
  name_field = "Name",
  ...
)

Arguments

genes Data frame containing start and stop in Mbp, strand (as ",-", "+", or NA), and Name.

minrow Minimum number of rows of genes in the plot

padding Proportion to pad with white space around the genes

colors Vectors of colors, used sequentially and then re-used.

scale_pos Factor by which to scale position (for example, to convert basepairs to Mbp)

start_field Character string with name of column containing the genes’ start positions.

stop_field Character string with name of column containing the genes’ stop positions.

strand_field Character string with name of column containing the genes’ strands. (The values of the corresponding field can be character strings "+" or "-", or numeric +1 or -1.)

name_field Character string with name of column containing the genes’ names.

...

Optional arguments passed to plot().

Value

None.

Hidden graphics parameters

Graphics parameters can be passed via .... For example, xlim to control the x-axis limits. These are not included as formal

Examples

genes <- data.frame(chr = c("6", "6", "6", "6", "6", "6", "6", "6"),
  start = c(139988753, 140680185, 141708118, 142234227, 142587862,
            143232344, 144398099, 144993835),
  stop = c(140041457, 140826797, 141773810, 142322981, 142702315,
           143260627, 144399821, 145076184),
  ...)

plot_genes(genes,
  minrow = 4,
  padding = 0.2,
  colors = c("black", "red3", "green4", "blue3", "orange"),
  scale_pos = 1,
  start_field = "start",
  stop_field = "stop",
  strand_field = "strand",
  name_field = "Name",
  ...
)
plot_genoprob

strand = c("-", "+", "-", "-", "-", NA, "+", "-"),
Name = c("Plcz1", "Gm30215", "Gm5724", "Slco1a5", "Abcc9",
"4930407I02Rik", "Gm31777", "Bcat1"),
stringsAsFactors=FALSE)

# use scale_pos=1e-6 because data in bp but we want the plot in Mbp
plot_genes(genes, xlim=c(140, 146), scale_pos=1e-6)

plot_genoprob

Plot genotype probabilities for one individual on one chromosome.

Description

Plot the genotype probabilities for one individual on one chromosome, as a heat map.

Usage

plot_genoprob(
    probs,
    map,
    ind = 1,
    chr = NULL,
    geno = NULL,
    color_scheme = c("gray", "viridis"),
    col = NULL,
    threshold = 0,
    swap_axes = FALSE,
    ...
)

## S3 method for class 'plot'
plot(x, ...)

Arguments

probs  Genotype probabilities (as produced by calc_genoprob()) or allele dosages (as produced by genoprob_to_alleleprob()).
map    Marker map (a list of vectors of marker positions).
ind    Individual to plot, either a numeric index or an ID.
chr    Selected chromosome to plot; a single character string.
geno   Optional vector of genotypes or alleles to be shown (vector of integers or character strings)
color_scheme Color scheme for the heatmap (ignored if col is provided).
col    Optional vector of colors for the heatmap.
threshold Threshold for genotype probabilities; only genotypes that achieve this value somewhere on the chromosome will be shown.
swap_axes

If TRUE, swap the axes, so that the genotypes are on the x-axis and the chromosome position is on the y-axis.

Additional graphics parameters passed to `graphics::image()`.

x

Genotype probabilities (as produced by `calc_genoprob()`) or allele dosages (as produced by `genoprob_to_alleleprob()`). (For the S3 type plot function, this has to be called x.)

Value

None.

Hidden graphics parameters

A number of graphics parameters can be passed via .... For example, hlines, hlines_col, hlines_lwd, and hlines_lty to control the horizontal grid lines. (Use hlines=NA to avoid plotting horizontal grid lines.) Similarly vlines, vlines_col, vlines_lwd, and vlines_lty for vertical grid lines. You can also use many standard graphics parameters like xlab and xlim. These are not included as formal parameters in order to avoid cluttering the function definition.

See Also

plot_genoprobcomp()

Examples

# load data and calculate genotype probabilities
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
iron <- iron[,"2"] # subset to chr 2
map <- insert_pseudomarkers(iron$gmap, step=1)
pr <- calc_genoprob(iron, map, error_prob=0.002)

# plot the probabilities for the individual labeled "262"
# (white = 0, black = 1)
plot_genoprob(pr, map, ind="262")

# change the x-axis label
plot_genoprob(pr, map, ind="262", xlab="Position (cM)")

# swap the axes so that the chromosome runs vertically
plot_genoprob(pr, map, ind="262", swap_axes=TRUE, ylab="Position (cM)")

# This is more interesting for a Diversity Outbred mouse example
## Not run:
file <- paste0("https://raw.githubusercontent.com/rqtl/",
"qtl2data/main/DOex/DOex.zip")
DOex <- read_cross2(file)
# subset to chr 2 and X and individuals labeled "232" and "256"
DOex <- DOex[c("232", "256"), c("2", "X")]
pr <- calc_genoprob(DOex, error_prob=0.002)
# plot individual "256" on chr 2 (default is to pick first chr in the probs)
plot_genoprob(pr, DOex$pmap, ind="256")
# omit states that never have probability >= 0.5
plot_genoprob(pr, DOex$pmap, ind="256", threshold=0.05)

# X chr male 232: just show the AY-HY genotype probabilities
plot_genoprob(pr, DOex$pmap, ind="232", chr="X", geno=paste0(LETTERS[1:8], "Y"))
# could also indicate genotypes by number
plot_genoprob(pr, DOex$pmap, ind="232", chr="X", geno=37:44)
# and can use negative indexes
plot_genoprob(pr, DOex$pmap, ind="232", chr="X", geno=-1:36))

# X chr female 256: just show the first 36 genotype probabilities
plot_genoprob(pr, DOex$pmap, ind="256", chr="X", geno=1:36)
# again, can give threshold to omit genotypes whose probabilities never reach that threshold
plot_genoprob(pr, DOex$pmap, ind="256", chr="X", geno=1:36, threshold=0.5)

# can also look at the allele dosages
apr <- genoprob_to_alleleprob(pr)
plot_genoprob(apr, DOex$pmap, ind="232")

## End(Not run)

---

**plot_genoprobcomp**

Plot comparison of two sets of genotype probabilities

---

**Description**

Plot a comparison of two sets of genotype probabilities for one individual on one chromosome, as a heat map.

**Usage**

```r
plot_genoprobcomp(
   probs1,
   probs2,
   map,
   ind = 1,
   chr = NULL,
   geno = NULL,
   threshold = 0,
   n_colors = 256,
   swap_axes = FALSE,
   ...
)
```
Arguments

probs1  Genotype probabilities (as produced by `calc_genoprob()`) or allele dosages (as produced by `genoprob_to_alleleprob()`).
probs2  A second set of genotype probabilities, just like `probs1`.
map    Marker map (a list of vectors of marker positions).
ind    Individual to plot, either a numeric index or an ID.
chr    Selected chromosome to plot; a single character string.
geno   Optional vector of genotypes or alleles to be shown (vector of integers or character strings).
threshold Threshold for genotype probabilities; only genotypes that achieve this value somewhere on the chromosome (in one or the other set of probabilities) will be shown.
n_colors Number of colors in each color scale.
swap_axes If TRUE, swap the axes, so that the genotypes are on the x-axis and the chromosome position is on the y-axis.
...    Additional graphics parameters passed to `graphics::image()`.

Details

We plot the first set of probabilities in the range white to blue and the second set in the range white to red and attempt to combine them, for colors that are white, some amount of blue or red, or where both are large something like blackish purple.

Value

None.

See Also

`plot_genoprob()`

Examples

```r
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
iron <- iron[228,"1"]  # subset to one individual on chr 1
map <- insert_pseudomarkers(iron$gmap, step=5)

# introduce genotype error and look at difference in genotype probabilities
pr_ne <- calc_genoprob(iron, map, error_prob=0.002)
iron$geno[[1]][[1,2]] <- 3
pr_e <- calc_genoprob(iron, map, error_prob=0.002)

# image of probabilities + comparison
par(mfrow=c(3,1))
plot_genoprob(pr_ne, map, main="No error")
plot_genoprob(pr_e, map, main="With an error")
```
plot_lodpeaks

plot_genoprobcomp(pr_ne, pr_e, map, main="Comparison")

plot_lodpeaks  Plot LOD scores vs QTL peak locations

Description

Create a scatterplot of LOD scores vs QTL peak locations (possibly with intervals) for multiple traits.

Usage

plot_lodpeaks(peaks, map, chr = NULL, gap = NULL, intervals = FALSE, ...)

Arguments

- **peaks**: Data frame such as that produced by `find_peaks()` containing columns `chr`, `pos`, `lodindex`, and `lodcolumn`. May also contain columns `ci_lo` and `ci_hi`, in which case intervals will be plotted.
- **map**: Marker map, used to get chromosome lengths (and start and end positions).
- **chr**: Selected chromosomes to plot; a vector of character strings.
- **gap**: Gap between chromosomes. The default is 1% of the total genome length.
- **intervals**: If TRUE and peaks contains QTL intervals, plot the intervals.
- **...**: Additional graphics parameters

Value

None.

Hidden graphics parameters

A number of graphics parameters can be passed via ... For example, bgcolor to control the background color and altbgcolor to control the background color on alternate chromosomes. These are not included as formal parameters in order to avoid cluttering the function definition.

See Also

`find_peaks()`, `plot_peaks()`
Examples

```r
# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)

# perform genome scan
out <- scan1(probs, pheno, addcovar=covar, Xcovar=Xcovar)

# find peaks above lod=3.5 (and calculate 1.5-LOD support intervals)
peaks <- find_peaks(out, map, threshold=3.5, drop=1.5)

plot_lodpeaks(peaks, map)
```

---

**plot_onegeno**

Plot one individual’s genome-wide genotypes

---

**Description**

Plot one individual’s genome-wide genotypes

**Usage**

```r
plot_onegeno(
    geno,
    map,
    ind = 1,
    chr = NULL,
    col = NULL,
    na_col = "white",
    swap_axes = FALSE,
    border = "black",
    shift = FALSE,
    chrwidth = 0.5,
    ...
)
```
**plot_onegeno**

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>geno</code></td>
<td>Imputed phase-known genotypes, as a list of matrices (as produced by <code>maxmarg()</code> or a list of three-dimensional arrays (as produced by <code>guess_phase()</code>).</td>
</tr>
<tr>
<td><code>map</code></td>
<td>Marker map (a list of vectors of marker positions).</td>
</tr>
<tr>
<td><code>ind</code></td>
<td>Individual to plot, either a numeric index or an ID.</td>
</tr>
<tr>
<td><code>chr</code></td>
<td>Selected chromosomes to plot; a vector of character strings.</td>
</tr>
<tr>
<td><code>col</code></td>
<td>Vector of colors for the different genotypes.</td>
</tr>
<tr>
<td><code>na_col</code></td>
<td>Color for missing segments.</td>
</tr>
<tr>
<td><code>swap_axes</code></td>
<td>If TRUE, swap the axes, so that the chromosomes run horizontally.</td>
</tr>
<tr>
<td><code>border</code></td>
<td>Color of outer border around chromosome rectangles.</td>
</tr>
<tr>
<td><code>shift</code></td>
<td>If TRUE, shift the chromosomes so they all start at 0.</td>
</tr>
<tr>
<td><code>chrwidth</code></td>
<td>Total width of rectangles for each chromosome, as a fraction of the distance between them.</td>
</tr>
</tbody>
</table>

... Additional graphics parameters

Value

None.

Hidden graphics parameters

A number of graphics parameters can be passed via ... For example, `bgcolor` to control the background color. These are not included as formal parameters in order to avoid cluttering the function definition.

Examples

```r
# load data and calculate genotype probabilities
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
iron <- iron["146", ] # subset to individual 146
map <- insert_pseudomarkers(iron$gmap, step=1)
pr <- calc_genoprob(iron, map, error_prob=0.002)

# infer genotypes, as those with maximal marginal probability
m <- maxmarg(pr)

# guess phase
ph <- guess_phase(iron, m)

# plot phased genotypes
plot_onegeno(ph, map, shift=TRUE, col=c("slateblue", "Orchid"))

# this is more interesting for Diversity Outbred mouse data
## Not run:
file <- paste0("https://raw.githubusercontent.com/rqtl/",
"qtl2data/main/DOex/DOex.zip")
DOex <- read_cross2(file)
```
# subset to individuals labeled "232" and "256"
DOex <- DOex[c("232", "256"), ]
pr <- calc_genoprob(DOex, error_prob=0.002)

# infer genotypes, as those with maximal marginal probability
m <- maxmarg(pr, minprob=0.5)
# guess phase
ph <- guess_phase(DOex, m)

# plot phased genotypes
plot_onegeno(ph, DOex$gmap, shift=TRUE)
plot_onegeno(ph, DOex$gmap, ind="256", shift=TRUE)

## End(Not run)

---

### plot_peaks

*Plot QTL peak locations*

**Description**

Plot QTL peak locations (possibly with intervals) for multiple traits.

**Usage**

```r
plot_peaks(
  peaks,
  map,
  chr = NULL,
  tick_height = 0.3,
  gap = NULL,
  lod_labels = FALSE,
  ...
)
```

**Arguments**

- **peaks**: Data frame such as that produced by `find_peaks()` containing columns chr, pos, lodindex, and lodcolumn. May also contain columns ci_lo and ci_hi, in which case intervals will be plotted.
- **map**: Marker map, used to get chromosome lengths (and start and end positions).
- **chr**: Selected chromosomes to plot; a vector of character strings.
- **tick_height**: Height of tick marks at the peaks (a number between 0 and 1).
- **gap**: Gap between chromosomes. The default is 1% of the total genome length.
plot_peaks

lod_labels  If TRUE, plot LOD scores near the intervals. Uses three hidden graphics parameters, label_gap (distance between CI and LOD text label), label_left (vector that indicates whether the labels should go on the left side; TRUE=on left, FALSE=on right, NA=put into larger gap on that chromosome), and label_cex that controls the size of these labels

Additional graphics parameters

Value

None.

Hidden graphics parameters

A number of graphics parameters can be passed via .... For example, bgcolor to control the background color and althbgcolor to control the background color on alternate chromosomes. These are not included as formal parameters in order to avoid cluttering the function definition.

See Also

find_peaks(), plot_lodpeaks()

Examples

# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)

# perform genome scan
out <- scan1(probs, pheno, addcovar=covar, Xcovar=Xcovar)

# find peaks above lod=3.5 (and calculate 1.5-LOD support intervals)
peaks <- find_peaks(out, map, threshold=3.5, drop=1.5)

plot_peaks(peaks, map)

# show LOD scores
plot_peaks(peaks, map, lod_labels=TRUE)

# show LOD scores, controlling whether they go on the left or right
plot_peaks(peaks, map, lod_labels=TRUE, label_left=c(TRUE, TRUE, TRUE, FALSE, TRUE, FALSE))

plot_pxg

Plot phenotype vs genotype

Description

Plot phenotype vs genotype for a single putative QTL and a single phenotype.

Usage

plot_pxg(
  geno,
  pheno,
  sort = TRUE,
  SEMult = NULL,
  pooledSD = TRUE,
  swap_axes = FALSE,
  jitter = 0.2,
  force_labels = TRUE,
  alternate_labels = FALSE,
  omit_points = FALSE,
  ...
)

Arguments

geno Vector of genotypes, for example as produced by maxmarg() with specific chr and pos.
pheno Vector of phenotypes.
sort If TRUE, sort genotypes from largest to smallest.
SEmult If specified, interval estimates of the within-group averages will be displayed, as mean +/- SE * SEMult.
pooledSD If TRUE and SEMult is specified, calculated a pooled within-group SD. Otherwise, get separate estimates of the within-group SD for each group.
swap_axes If TRUE, swap the axes, so that the genotypes are on the y-axis and the phenotype is on the x-axis.
jitter Amount to jitter the points horizontally, if a vector of length > 0, it is taken to be the actual jitter amounts (with values between -0.5 and 0.5).
force_labels If TRUE, force all genotype labels to be shown.
alternate_labels If TRUE, place genotype labels in two rows
omit_points If TRUE, omit the points, just plotting the averages (and, potentially, the +/- SE intervals).
... Additional graphics parameters, passed to plot().
Value

(Invisibly) A matrix with rows being the genotype groups and columns for the means and (if SEmult is specified) the SEs.

Hidden graphics parameters

A number of graphics parameters can be passed via .... For example, bgcolor to control the background color, and seg_width, seg_lwd, and seg_col to control the lines at the confidence intervals. Further, hlines, hlines_col, hlines_lwd, and hlines_lty to control the horizontal grid lines. (Use hlines=NA to avoid plotting horizontal grid lines.) Similarly vlines, vlines_col, vlines_lwd, and vlines_lty for vertical grid lines. These are not included as formal parameters in order to avoid cluttering the function definition.

See Also

plot_coef()

Examples

# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# inferred genotype at a 28.6 cM on chr 16
gen1 <- maxmarg(probs, map, chr=16, pos=28.6, return_char=TRUE)

# plot phenotype vs genotype
plot_pxg(gen1, log10(iron$pheno[,1]), ylab=expression(log[10](Liver)))

# include +/- 2 SE intervals
plot_pxg(gen1, log10(iron$pheno[,1]), ylab=expression(log[10](Liver)),
SEmult=2)

# plot just the means
plot_pxg(gen1, log10(iron$pheno[,1]), ylab=expression(log[10](Liver)),
omit_points=TRUE)

# plot just the means +/- 2 SEs
plot_pxg(gen1, log10(iron$pheno[,1]), ylab=expression(log[10](Liver)),
omit_points=TRUE, SEmult=2)
plot_scan1  

Plot a genome scan

Description

Plot LOD curves for a genome scan

Usage

```r
plot_scan1(x, map, lodcolumn = 1, chr = NULL, add = FALSE, gap = NULL, ...)
```  
```r
## S3 method for class 'scan1'
plot(x, map, lodcolumn = 1, chr = NULL, add = FALSE, gap = NULL, ...)
```

Arguments

- `x`: An object of class "scan1", as output by `scan1()`.  
- `map`: A list of vectors of marker positions, as produced by `insert_pseudomarkers()`.  
- `lodcolumn`: LOD score column to plot (a numeric index, or a character string for a column name). Only one value allowed.  
- `chr`: Selected chromosomes to plot; a vector of character strings.  
- `add`: If TRUE, add to current plot (must have same map and chromosomes).  
- `gap`: Gap between chromosomes. The default is 1% of the total genome length.  
- `...`: Additional graphics parameters.

Value

None.

Hidden graphics parameters

A number of graphics parameters can be passed via `...`. For example, `bgcolor` to control the background color and `altbgcolor` to control the background color on alternate chromosomes. `col` controls the color of lines/curves; `altcol` can be used if you want alternative chromosomes in different colors. These are not included as formal parameters in order to avoid cluttering the function definition.

See Also

- `plot_coef()`, `plot_coefCC()`, `plot_snpasso()`
Examples

# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)

# perform genome scan
out <- scan1(probs, pheno, addcovar=covar, Xcovar=Xcovar)

# plot the results for selected chromosomes
ylim <- c(0, maxlod(out)*1.02) # need to strip class to get overall max LOD
chr <- c(2,7,8,9,15,16)
plot(out, map, chr=chr, ylim=ylim)
plot(out, map, lodcolumn=2, chr=chr, col="violetred", add=TRUE)
legend("topleft", lwd=2, col=c("darkslateblue", "violetred"), colnames(out),
bg="gray90")

# plot just one chromosome
plot(out, map, chr=8, ylim=ylim)
plot(out, map, chr=8, lodcolumn=2, col="violetred", add=TRUE)

# lodcolumn can also be a column name
plot(out, map, lodcolumn="liver", ylim=ylim)
plot(out, map, lodcolumn="spleen", col="violetred", add=TRUE)

---

plot_sdp

plot strain distribution patterns for SNPs

Description

plot the strain distribution patterns of SNPs using tracks of tick-marks for each founder strain

Usage

plot_sdp(pos, sdp, strain_labels = names(qtl2::CCcolors), ...)

Arguments

pos vector of SNP positions
plot_snpasso

```r
sdp vector of strain distribution patterns (as integers)
strain_labels names of the strains
... additional graphic arguments
```

Details

Additional arguments, such as `xlab`, `ylab`, `xlim`, and `main`, are passed via `...`; also `bgcolor` to control the color of the background, and `col` and `lwd` to control the color and thickness of the tick marks.

Value

None.

See Also

`calc_sdp()`, `invert_sdp()`

Examples

```r
n_tick <- 50
plot_sdp(runif(n_tick, 0, 100), sample(0:255, n_tick, replace=TRUE))
```

plot_snpasso  

**Plot SNP associations**

Description

Plot SNP associations, with possible expansion from distinct snps to all snps.

Usage

```r
plot_snpasso(
    scan1output,
    snpinfo,
    genes = NULL,
    lodcolumn = 1,
    show_all_snps = TRUE,
    chr = NULL,
    add = FALSE,
    drop_hilit = NA,
    col_hilit = "violetred",
    col = "darkslateblue",
    gap = NULL,
    minlod = 0,
    sdp_panel = FALSE,
    strain_labels = names(qtl2::CCcolors),
    ...
)
```
plot_snpasso

Arguments

scan1output Output of scan1() using SNP probabilities derived by genoprob_to_snpprob().
snpinfo Data frame with SNP information with the following columns (the last three are generally derived from with index_snps()):
  • chr - Character string or factor with chromosome
  • pos - Position (in same units as in the "map" attribute in genoprobs.
  • sdp - Strain distribution pattern: an integer, between 1 and \(2^n - 2\) where \(n\) is the number of strains, whose binary encoding indicates the founder genotypes
  • snp - Character string with SNP identifier (if missing, the rownames are used).
  • index - Indices that indicate equivalent groups of SNPs.
  • intervals - Indexes that indicate which marker intervals the SNPs reside.
  • on_map - Indicate whether SNP coincides with a marker in the genoprobs
genes Optional data frame containing gene information for the region, with columns start and stop in Mbp, strand (as "+", "+", or NA), and Name. If included, a two-panel plot is produced, with SNP associations above and gene locations below.
lodcolumn LOD score column to plot (a numeric index, or a character string for a column name). Only one value allowed.
show_all_snps If TRUE, expand to show all SNPs.
chr Vector of character strings with chromosome IDs to plot.
add If TRUE, add to current plot (must have same map and chromosomes).
drop_hilit SNPs with LOD score within this amount of the maximum SNP association will be highlighted.
col_hilit Color of highlighted points
col Color of other points
gap Gap between chromosomes. The default is 1% of the total genome length.
minlod Minimum LOD to display. (Mostly for GWAS, in which case using minlod=1 will greatly increase the plotting speed, since the vast majority of points would be omitted.
sdp_panel Include a panel with the strain distribution patterns for the highlighted SNPs
strain_labels Labels for the strains, if sdp_panel=TRUE.
... Additional graphics parameters.

Value

None.
Hidden graphics parameters

A number of graphics parameters can be passed via .... For example, bgcolor to control
the background color, altbgcolor to control the background color on alternate chromosomes, altcol
to control the point color on alternate chromosomes, cex for character expansion for the points
(default 0.5), pch for the plotting character for the points (default 16), and ylim for y-axis limits. If
you are including genes and/or SDP panels, you can use panel_prop to control the relative heights
of the panels, from top to bottom.

See Also

plot_scan1(), plot_coef(), plot_coefCC()

Examples

## Not run:
# load example DO data from web
file <- paste0("https://raw.githubusercontent.com/rqtl",
"/qtl2data/main/DOex/DOex.zip")
DOex <- read_cross2(file)

# subset to chr 2
DOex <- DOex[,2]

# calculate genotype probabilities and convert to allele probabilities
pr <- calc_genoprob(DOex, error_prob=0.002)
apr <- genoprob_to_alleleprob(pr)

# query function for grabbing info about variants in region
snp_dbfile <- system.file("extdata", "cc_variants_small.sqlite", package="qtl2")
query_variants <- create_variant_query_func(snp_dbfile)

# SNP association scan
out_snps <- scan1snps(apr, DOex$pmap, DOex$pheno, query_func=query_variants,
chr=2, start=97, end=98, keep_all_snps=TRUE)

# plot results
plot_snpasso(out_snps$lod, out_snps$snpinfo)

# can also just type plot()
plot(out_snps$lod, out_snps$snpinfo)

# plot just subset of distinct SNPs
plot(out_snps$lod, out_snps$snpinfo, show_all_snps=FALSE)

# highlight the top snps (with LOD within 1.5 of max)
plot(out_snps$lod, out_snps$snpinfo, drop_hilit=1.5)

# query function for finding genes in region
gene_dbfile <- system.file("extdata", "mouse_genes_small.sqlite", package="qtl2")
query_genes <- create_gene_query_func(gene_dbfile)
genes <- query_genes(2, 97, 98)
# plot SNP association results with gene locations
plot(out_snps$lod, out_snps$snpinfo, drop_hilit=1.5, genes=genes)

# plot SNP association results with genes plus SDPs of highlighted SNPs
plot(out_snps$lod, out_snps$snpinfo, drop_hilit=2, genes=genes, sdp_panel=TRUE)

## End(Not run)

---

**predict_snpgeno**  
Predict SNP genotypes

**Description**  
Predict SNP genotypes in a multiparent population from inferred genotypes plus founder strains’ SNP alleles.

**Usage**  
predict_snpgeno(cross, geno, cores = 1)

**Arguments**
- **cross** Object of class "cross2". For details, see the R/qtl2 developer guide.
- **geno** Imputed genotypes, as a list of matrices, as from `maxmarg()`.
- **cores** Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`. Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.

**Value**
A list of matrices with inferred SNP genotypes, coded 1/2/3.

**See Also**
`maxmarg()`, `viterbi()`, `calc_errorlod()`

**Examples**
```r
## Not run:
# load example data and calculate genotype probabilities
file <- paste0("https://raw.githubusercontent.com/rqtl/",
"qtl2data/main/DOex/DOex.zip")
DOex <- read_cross2(file)
probs <- calc_genoprob(DOex, error_prob=0.002)

# inferred genotypes
m <- maxmarg(probs, minprob=0.5)
```
# inferred SNP genotypes
inferg <- predict_snpgeno(DOex, m)

## End(Not run)

---

**print.cross2**

*Print a cross2 object*

**Description**

Print a summary of a cross2 object

**Usage**

```r
## S3 method for class 'cross2'
print(x, ...)
```

**Arguments**

- **x**: An object of class "cross2", as output by `read_cross2()`. For details, see the R/qtl2 developer guide.
- **...**: Ignored.

**Value**

None.

---

**print.summary.scan1perm**

*Print summary of scan1perm permutations*

**Description**

Print summary of scan1perm permutations

**Usage**

```r
## S3 method for class 'summary.scan1perm'
print(x, digits = 3, ...)
```

**Arguments**

- **x**: Object of class "summary.scan1perm", as produced by `summary_scan1perm()`.
- **digits**: Number of digits in printing significance thresholds; passed to `base::print()`.
- **...**: Ignored.
Details

This is to go with `summary_scan1perm()`, so that the summary output is printed in a nice format. Generally not called directly, but it can be in order to control the number of digits that appear.

Value

Invisibly returns the input, \( x \).

Examples

```R
# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)

# permutations with genome scan (just 3 replicates, for illustration)
operm <- scan1perm(probs, pheno, addcovar=covar, Xcovar=Xcovar,
n_perm=3)

print( summary(operm, alpha=c(0.20, 0.05)), digits=8 )
```

---

**probs_to_grid**

Subset genotype probability array to pseudomarkers on a grid

**Description**

Subset genotype probability array (from `calc_genoprob()`) to a grid of pseudomarkers along each chromosome.

**Usage**

```R
probs_to_grid(probs, grid)
```

**Arguments**

- **probs**: Genotype probabilities as output from `calc_genoprob()` with stepwidth="fixed".
- **grid**: List of logical vectors that indicate which positions are on the grid and should be retained.
Details

This only works if `calc_genoprob()` was run with `stepwidth="fixed"`, so that the genotype probabilities were calculated at a grid of markers/pseudomarkers. When this is the case, we omit all but the probabilities on this grid. Use `calc_grid()` to find the grid positions.

Value

An object of class "calc_genoprob", like the input, subset to just include pseudomarkers along a grid. See `calc_genoprob()`.

See Also

calc_grid(), map_to_grid()

Examples

```r
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
map_w_pmar <- insert_pseudomarkers(grav2$gmap, step=1)
probs <- calc_genoprob(grav2, map_w_pmar, error_prob=0.002)
sapply(probs, dim)
grid <- calc_grid(grav2$gmap, step=1)
probs_sub <- probs_to_grid(probs, grid)
sapply(probs_sub, dim)
```

---

pull_genoprobint  

Pull genotype probabilities for an interval

Description

Pull out the genotype probabilities for a given genomic interval

Usage

`pull_genoprobint(genoprosbs, map, chr, interval)`

Arguments

- `genoprosbs`: Genotype probabilities as calculated by `calc_genoprob()`.
- `map`: The marker map for the genotype probabilities.
- `chr`: Chromosome ID (single character sting).
- `interval`: Interval (pair of numbers).

Value

A list containing a single 3d array of genotype probabilities, like the input genoprosbs but for the designated interval.
See Also

`find_marker()`, `pull_genoprobpos()`

Examples

```r
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

gmap <- insert_pseudomarkers(iron$gmap, step=1)
pr <- calc_genoprob(iron, gmap, error_prob=0.002)
pr_sub <- pull_genoprobint(pr, gmap, "8", c(25, 35))
```

---

**pull_genoprobpos**

*Pull genotype probabilities for a particular position*

**Description**

Pull out the genotype probabilities for a particular position (by name)

**Usage**

```r
pull_genoprobpos(genoprobs, map = NULL, chr = NULL, pos = NULL, marker = NULL)
```

**Arguments**

- `genoprobs` Genotype probabilities as calculated by `calc_genoprob()`.
- `map` A map object: a list (corresponding to chromosomes) of vectors of marker positions. Can also be a `snpinfo` object (data frame with columns `chr` and `pos`; marker names taken from column `snp` or if that doesn’t exist from the row names).
- `chr` A chromosome ID
- `pos` A numeric position
- `marker` A single character string with the name of the position to pull out.

**Details**

Provide either a marker/pseudomarker name (with the argument `marker`) or all of `map`, `chr`, and `pos`.

**Value**

A matrix of genotype probabilities for the specified position.

**See Also**

`find_marker()`, `fit1()`, `pull_genoprobint()`
pull_markers

Drop all but a specified set of markers

Description

Drop all markers from a cross2 object expect those in a specified vector.

Usage

pull_markers(cross, markers)

Arguments

cross   Object of class "cross2". For details, see the R/qtl2 developer guide.
markers A vector of marker names.

Value

The input cross with only the specified markers.

See Also

drop_markers(), drop_nullmarkers()

Examples

grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
markers2drop <- c("BH.342C/347L-Col", "GH.94L", "EG.357C/359L-Col", "CD.245L", "ANL2")
grav2_rev <- pull_markers(grav2, markers2drop)
qtl2version

Installed version of R/qtl2

Description
Get installed version of R/qtl2

Usage
qtl2version()

Value
A character string with the installed version of the R/qtl2 package.

Examples
qtl2version()

rbind.calc_genoprob
Join genotype probabilities for different individuals

Description
Join multiple genotype probability objects, as produced by calc_genoprob(), for the same set of markers and genotypes but for different individuals.

Usage
## S3 method for class 'calc_genoprob'
rbind(...)  

Arguments
...
Genotype probability objects as produced by calc_genoprob(). Must have the same set of markers and genotypes.

Value
An object of class "calc_genoprob", like the input; see calc_genoprob().

See Also
cbind.calc_genoprob()
Examples

```r
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
map <- insert_pseudomarkers(grav2$gmap, step=1)
probsA <- calc_genoprob(grav2[1:5,], map, error_prob=0.002)
probsB <- calc_genoprob(grav2[6:12,], map, error_prob=0.002)
probs <- rbind(probsA, probsB)
```

---

**rbind.scan1**  
*Join genome scan results for different chromosomes.*

**Description**

Join multiple scan1() results for different chromosomes; must have the same set of lod score column.

**Usage**

```r
## S3 method for class 'scan1'
rbind(...)
```

**Arguments**

```r
...  
```

Genome scan objects of class "scan1", as produced by scan1(). Must have the same lod score columns.

**Details**

If components addcovar, Xcovar, intcovar, weights, sample_size do not match between objects, we omit this information.

If hsq present, we simply rbind() the contents.

**Value**

An object of class "scan1", like the inputs, but with the results for different sets of chromosomes combined.

**See Also**

cbind.scan1(), scan1()
```
Examples

grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
map <- insert_pseudomarkers(grav2$gmap, step=1)
probs <- calc_genoprob(grav2, map, error_prob=0.002)
phe <- grav2$pheno[,1,drop=FALSE]

out1 <- scan1(probs[,1], phe) # chr 1
out2 <- scan1(probs[,5], phe) # chr 5
out <- rbind(out1, out2)
```

---

**rbind.scan1perm**  
*Combine data from scan1perm objects*

**Description**
Row-bind multiple scan1perm objects with the same set of columns

**Usage**

```r
## S3 method for class 'scan1perm'
rbind(...)
## S3 method for class 'scan1perm'
c(...)
```

**Arguments**

...  
A set of permutation results from `scan1perm()` (objects of class "scan1perm"). They must have the same set of columns. If any include autosome/X chromosome-specific permutations, they must all be such.

**Details**

The aim of this function is to concatenate the results from multiple runs of a permutation test with `scan1perm()`, to assist in the case that such permutations are done on multiple processors in parallel.

**Value**

The combined row-binded input, as an object of class "scan1perm"; see `scan1perm()`.

**See Also**

`cbind.scan1perm()`, `scan1perm()`, `scan1()`
Examples

# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)

# permutations with genome scan (just 3 replicates, for illustration)
operm1 <- scan1perm(probs, pheno, addcovar=covar, Xcovar=Xcovar, n_perm=3)
operm2 <- scan1perm(probs, pheno, addcovar=covar, Xcovar=Xcovar, n_perm=3)

operm <- rbind(operm1, operm2)

---

rbind.sim_geneno

Join genotype imputations for different individuals

Description

Join multiple genotype imputation objects, as produced by sim_geneno(), for the same set of markers but for different individuals.

Usage

## S3 method for class "sim_geneno"
rbind(...)

Arguments

... Genotype imputation objects as produced by sim_geneno(). Must have the same set of markers and genotypes.

Value

An object of class "sim_geneno", like the input; see sim_geneno().

See Also

cbind.sim_geneno(), sim_geneno()
Examples

```
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
map <- insert_pseudomarkers(grav2$gmap, step=1)
drawsA <- sim_geno(grav2[1:5,], map, error_prob=0.002, n_draws=4)
drawsB <- sim_geno(grav2[6:12,], map, error_prob=0.002, n_draws=4)
draws <- rbind(drawsA, drawsB)
```

---

**rbind.viterbi** *Join Viterbi results for different individuals*

**Description**

Join multiple imputed genotype objects, as produced by `viterbi()`, for the same set of markers but for different individuals.

**Usage**

```r
## S3 method for class 'viterbi'
rbind(...)
```

**Arguments**

...  
Imputed genotype objects as produced by `viterbi()`. Must have the same set of markers.

**Value**

An object of class "viterbi", like the input; see `viterbi()`.

**See Also**

`cbind.viterbi()`, `viterbi()`

**Examples**

```
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
map <- insert_pseudomarkers(grav2$gmap, step=1)
gA <- viterbi(grav2[1:5,], map, error_prob=0.002)
gB <- viterbi(grav2[6:12,], map, error_prob=0.002)
g <- rbind(gA, gB)
```
read_cross2  
**Read QTL data from files**

**Description**
Read QTL data from a set of files

**Usage**
`read_cross2(file, quiet = TRUE)`

**Arguments**
- `file`  
  Character string with path to the YAML or JSON file containing all of the control information. This could instead be a zip file containing all of the data files, in which case the contents are unzipped to a temporary directory and then read.
- `quiet`  
  If FALSE, print progress messages.

**Details**
A control file in YAML or JSON format contains information about basic parameters as well as the names of the series of data files to be read. See the sample data files and the vignette describing the input file format.

**Value**
Object of class "cross2". For details, see the R/qtl2 developer guide.

**See Also**
- `read_pheno()`, `write_control_file()`.

**Examples**
```r
## Not run:
yaml_file <- "https://kbroman.org/qtl2/assets/sampledata/grav2/grav2.yaml"
grav2 <- read_cross2(yaml_file)

## End(Not run)
zip_file <- system.file("extdata", "grav2.zip", package="qtl2")
grav2 <- read_cross2(zip_file)
```
read_csv

Read a csv file via data.table::fread() using a particular set of options, including the ability to transpose the result.

Usage

read_csv(
  filename,
  sep = "","n",
  na.strings = c("NA", "-"),
  comment.char = "#",
  transpose = FALSE,
  rownames_included = TRUE
)

Arguments

filename Name of input file
sep Field separator
na.strings Missing value codes
comment.char Comment character; rest of line after this character is ignored
transpose If TRUE, transpose the result
rownames_included If TRUE, the first column is taken to be row names.

Details

Initial two lines can contain comments with number of rows and columns. Number of columns includes an ID column; number of rows does not include the header row.
The first column is taken to be a set of row names

Value

Data frame

See Also

read_csv_numer()

Examples

## Not run: mydata <- read_csv("myfile.csv", transpose=TRUE)
**read_csv_numer**  
Read a csv file that has numeric columns

**Description**

Read a csv file via `data.table::fread()` using a particular set of options, including the ability to transpose the result. This version assumes that the contents other than the first column and the header row are strictly numeric.

**Usage**

```r
read_csv_numer(
    filename,
    sep = ",",
    na.strings = c("NA", ":"),
    comment.char = "#",
    transpose = FALSE,
    rownames_included = TRUE
)
```

**Arguments**

- `filename`  Name of input file
- `sep`  Field separator
- `na.strings`  Missing value codes
- `comment.char`  Comment character; rest of line after this character is ignored
- `transpose`  If TRUE, transpose the result
- `rownames_included`  If TRUE, the first column is taken to be row names.

**Details**

Initial two lines can contain comments with number of rows and columns. Number of columns includes an ID column; number of rows does not include the header row.

The first column is taken to be a set of row names

**Value**

Data frame

**See Also**

- `read_csv()`

**Examples**

```r
## Not run: mydata <- read_csv_numer("myfile.csv", transpose=TRUE)
```
Description

Read phenotype data from a CSV file (and, optionally, phenotype covariate data from a separate CSV file). The CSV files may be contained in zip files, separately or together.

Usage

read_pheno(
  file,
  phenocovarfile = NULL,
  sep = ",",
  na.strings = c("-", "NA"),
  comment.char = "#",
  transpose = FALSE,
  quiet = TRUE
)

Arguments

file Character string with path to the phenotype data file (or a zip file containing both the phenotype and phenotype covariate files).

phenocovarfile Character string with path to the phenotype covariate file. This can be a separate CSV or zip file; if a zip file, it must contain exactly one CSV file. Alternatively, if the file argument indicates a zip file that contains two files (phenotypes and phenotype covariates), then this phenocovarfile argument must indicate the base name for the phenotype covariate file.

sep the field separator character

na.strings a character vector of strings which are to be interpreted as NA values.

comment.char A character vector of length one containing a single character to denote comments within the CSV files.

transpose If TRUE, the phenotype data will be transposed. The phenotype covariate information is never transposed.

quiet If FALSE, print progress messages.

Value

Either a matrix of phenotype data, or a list containing pheno (phenotype matrix) and phenocovar (phenotype covariate matrix).

See Also

Examples

## Not run:
```r
file <- paste0("https://raw.githubusercontent.com/rqtl/",
    "qtl2data/main/Gough/gough_pheno.csv")
phe <- read_pheno(file)

phecovfile <- paste0("https://raw.githubusercontent.com/rqtl/",
    "qtl2data/main/Gough/gough_phenocovar.csv")
phec_list <- read_pheno(file, phecovfile)
```  
## End(Not run)

---

recode_snps  
Recode SNPs by major allele

Description

For multi-parent populations with founder genotypes, recode the raw SNP genotypes so that 1 means homozygous for the major allele in the founders.

Usage

```
recode_snps(cross)
```

Arguments

- `cross`  
  Object of class "cross2". For details, see the R/qtl2 developer guide.

Value

The input cross object with the raw SNP genotypes recoded so that 1 is homozygous for the major alleles in the founders.

See Also

- `calc_raw_founder_maf()`, `calc_raw_maf()`

Examples

## Not run:
```r
# load example data and calculate genotype probabilities
file <- paste0("https://raw.githubusercontent.com/rqtl/",
    "qtl2data/main/DOex/DOex.zip")
DOex <- read_cross2(file)
DOex <- recode_snps(DOex)
```  
## End(Not run)
reduce_map_gaps

Reduce the lengths of gaps in a map

Description

Reduce the lengths of gaps in a map

Usage

reduce_map_gaps(map, min_gap = 50)

Arguments

map Genetic map as a list of vectors (each vector is a chromosome and contains the marker positions).
min_gap Minimum gap length to return.

Value

Input map with any gaps greater than min_gap reduced to min_gap.

See Also

find_map_gaps()

Examples

iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
rev_map <- reduce_map_gaps(iron$gmap, 30)

reduce_markers

Reduce markers to a subset of more-evenly-spaced ones

Description

Find the largest subset of markers such that no two adjacent markers are separated by less than some distance.
Usage

```r
reduce_markers(
  map,
  min_distance = 1,
  weights = NULL,
  max_batch = 10000,
  batch_distance_mult = 1,
  cores = 1
)
```

Arguments

- `map`: A list with each component being a vector with the marker positions for a chromosome.
- `min_distance`: Minimum distance between markers.
- `weights`: A (optional) list of weights on the markers; same size as `map`.
- `max_batch`: Maximum number of markers to consider in a batch.
- `batch_distance_mult`: If working with batches of markers, reduce `min_distance` by this multiple.
- `cores`: Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`.

Details

Uses a dynamic programming algorithm to find, for each chromosome, the subset of markers for which `max(weights)` is maximal, subject to the constraint that no two adjacent markers may be separated by more than `min_distance`.

The computation time for the algorithm grows like the square of the number of markers, like 1 sec for 10k markers but 30 sec for 50k markers. If the number of markers on a chromosome is greater than `max_batch`, the markers are split into batches and the algorithm applied to each batch with `min_distance` smaller by a factor `min_distance_mult`, and then merged together for one last pass.

Value

A list like the input `map`, but with the selected subset of markers.

References


Examples

```r
# read data
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))

# grab genetic map
```
replace_ids

```r
gmap <- grav2$gmap

# subset to markers that are >= 1 cM apart
gmap_sub <- reduce_markers(gmap, 1)

# drop all of the other markers from the cross
markers2keep <- unlist(lapply(gmap_sub, names))
grav2_sub <- pull_markers(grav2, markers2keep)
```

---

**Description**

Replace individual IDs

**Usage**

```r
replace_ids(x, ids)
```

**Arguments**

- `x` Object whose IDs will be replaced
- `ids` Vector of character strings with the new individual IDs, with the names being the original IDs.

**Value**

The input `x` object, but with individual IDs replaced.
Methods (by class)

- `replace_ids(cross2)`: Replace IDs in a "cross2" object
- `replace_ids(calc_genoprob)`: Replace IDs in output from `calc_genoprob()`
- `replace_ids(viterbi)`: Replace IDs in output from `viterbi()`
- `replace_ids(sim_geno)`: Replace IDs in output from `sim_geno()`
- `replace_ids(matrix)`: Replace IDs in a matrix
- `replace_ids(data.frame)`: Replace IDs in a data frame

Examples

```r
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
ids <- as.numeric(ind_ids(iron))

# replace the numeric IDs with IDs like "mouse003"
new_ids <- setNames( sprintf("mouse%03d", as.numeric(ids)), ids)

iron <- replace_ids(iron, new_ids)
```

---

scale_kinship

**Scale kinship matrix**

**Description**

Scale kinship matrix to be like a correlation matrix.

**Usage**

```r
scale_kinship(kinship)
```

**Arguments**

- `kinship` A kinship matrix, or a list of such in the case of the "leave one chromosome out" method, as calculated by `calc_kinship()`.

**Details**

We take \( c_{ij} = \frac{k_{ij}}{\sqrt{k_{ii}k_{jj}}} \)

**Value**

A matrix or list of matrices, as with the input, but with the matrices scaled to be like correlation matrices.
Examples

```r
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
map <- insert_pseudomarkers(grav2$gmap, step=1)
probs <- calc_genoprob(grav2, map, error_prob=0.002)
K <- calc_kinship(probs)
Ka <- scale_kinship(K)
```

---

**Genome scan with a single-QTL model**

**Description**

Genome scan with a single-QTL model by Haley-Knott regression or a linear mixed model, with possible allowance for covariates.

**Usage**

```r
scan1(
  genoprobs,
  pheno,
  kinship = NULL,
  addcovar = NULL,
  Xcovar = NULL,
  intcovar = NULL,
  weights = NULL,
  reml = TRUE,
  model = c("normal", "binary"),
  hsq = NULL,
  cores = 1,
  ...
)
```

**Arguments**

- `genoprobs`: Genotype probabilities as calculated by `calc_genoprob()`.
- `pheno`: A numeric matrix of phenotypes, individuals x phenotypes.
- `kinship`: Optional kinship matrix, or a list of kinship matrices (one per chromosome), in order to use the LOCO (leave one chromosome out) method.
- `addcovar`: An optional numeric matrix of additive covariates.
- `Xcovar`: An optional numeric matrix with additional additive covariates used for null hypothesis when scanning the X chromosome.
- `intcovar`: An numeric optional matrix of interactive covariates.
- `weights`: An optional numeric vector of positive weights for the individuals. As with the other inputs, it must have names for individual identifiers.
- `reml`: If kinship provided: if `reml=TRUE`, use REML; otherwise maximum likelihood.
model
Indicates whether to use a normal model (least squares) or binary model (logistic
regression) for the phenotype. If model="binary", the phenotypes must have
values in \([0, 1]\).

hsq
Considered only if kinship is provided, in which case this is taken as the as-
sumed value for the residual heritability. It should be a vector with length corre-
sponding to the number of columns in pheno, or (if kinship corresponds to a list
of LOCO kinship matrices) a matrix with dimension length(kinship) x ncol(pheno).

tol
is used as a tolerance value for linear regres-
sion by QR decomposition (in determining whether columns are linearly dependent on others and
should be omitted); default 1e-12.

intcovar_method
indicates whether to use a high-memory (but
potentially faster) method or a low-memory (and possibly slower) method, with values "highmem"
or "lowmem"; default "lowmem".

max_batch
indicates the maximum number of phenotypes to run
together; default is unlimited.

maxit
is the maximum number of iterations for con-
vergence of the iterative algorithm used when model=binary.

bintol
is used as a tolerance for con-
vergence for the iterative algorithm used when model=binary.

eta_max
is the maximum value for the "linear
predictor" in the case model="binary" (a bit of a technicality to avoid fitted values exactly at 0 or

If kinship is absent, Haley-Knott regression is performed. If kinship is provided, a linear mixed
model is used, with a polygenic effect estimated under the null hypothesis of no (major) QTL, and
then taken as fixed as known in the genome scan.

If kinship is a single matrix, then the hsq in the results is a vector of heritabilities (one value
for each phenotype). If kinship is a list (one matrix per chromosome), then hsq is a matrix,
chromosomes x phenotypes.

Value
An object of class "scan1": a matrix of LOD scores, positions x phenotypes. Also contains one or
more of the following attributes:

- sample_size - Vector of sample sizes used for each phenotype
scan1blup

- `hsq` - Included if kinship provided: A matrix of estimated heritabilities under the null hypothesis of no QTL. Columns are the phenotypes. If the "loco" method was used with `calc_kinship()` to calculate a list of kinship matrices, one per chromosome, the rows of `hsq` will be the heritabilities for the different chromosomes (well, leaving out each one). If `Xcovar` was not NULL, there will at least be an autosome and X chromosome row.

References


See Also

`scan1perm()`, `scan1coef()`, `cbind.scan1()`, `rbind.scan1()`, `scan1max()`

Examples

# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)

# perform genome scan
out <- scan1(probs, pheno, addcovar=covar, Xcovar=Xcovar)

# leave-one-chromosome-out kinship matrices
kinship <- calc_kinship(probs, "loco")

# genome scan with a linear mixed model
out_lmm <- scan1(probs, pheno, kinship, covar, Xcovar)
**Description**

Calculate BLUPs of QTL effects in scan along one chromosome, with a single-QTL model treating the QTL effects as random, with possible allowance for covariates and for a residual polygenic effect.

**Usage**

```r
scan1blup(
    genoprobs,
    pheno,
    kinship = NULL,
    addcovar = NULL,
    nullcovar = NULL,
    contrasts = NULL,
    se = FALSE,
    reml = TRUE,
    tol = 0.000000000001,
    cores = 1,
    quiet = TRUE
)
```

**Arguments**

- `genoprobs`: Genotype probabilities as calculated by `calc_genoprob()`.
- `pheno`: A numeric vector of phenotype values (just one phenotype, not a matrix of them).
- `kinship`: Optional kinship matrix, or a list of kinship matrices (one per chromosome), in order to use the LOCO (leave one chromosome out) method.
- `addcovar`: An optional numeric matrix of additive covariates.
- `nullcovar`: An optional numeric matrix of additional additive covariates that are used under the null hypothesis (of no QTL) but not under the alternative (with a QTL). This is needed for the X chromosome, where we might need sex as a additive covariate under the null hypothesis, but we wouldn’t want to include it under the alternative as it would be collinear with the QTL effects. Only used if `kinship` is provided but `hsq` is not, to get estimate of residual heritability.
- `contrasts`: An optional numeric matrix of genotype contrasts, size genotypes x genotypes. For an intercross, you might use `cbind(mu=c(1,0,0), a=c(-1, 0, 1), d=c(0, 1, 0))` to get mean, additive effect, and dominance effect. The default is the identity matrix.
- `se`: If TRUE, also calculate the standard errors.
- `reml`: If `reml=TRUE`, use REML to estimate variance components; otherwise maximum likelihood.
- `tol`: Tolerance value for convergence of linear mixed model fit.
- `cores`: Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`. Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.
- `quiet`: If FALSE, print message about number of cores used when multi-core.
Details
For each of the inputs, the row names are used as individual identifiers, to align individuals.
If kinship is provided, the linear mixed model accounts for a residual polygenic effect, with a the polygenic variance estimated under the null hypothesis of no (major) QTL, and then taken as fixed as known in the scan to estimate QTL effects.
If contrasts is provided, the genotype probability matrix, \( P \), is post-multiplied by the contrasts matrix, \( A \), prior to fitting the model. So we use \( P \cdot A \) as the \( X \) matrix in the model. One might view the rows of \( A^{-1} \) as the set of contrasts, as the estimated effects are the estimated genotype effects pre-multiplied by \( A^{-1} \).

Value
An object of class "scan1coef": a matrix of estimated regression coefficients, of dimension positions x number of effects. The number of effects is \( n_{\text{genotypes}} + n_{\text{addcovar}} + (n_{\text{genotypes}}-1)\cdot n_{\text{intcovar}} \).
May also contain the following attributes:
- \text{SE} - Present if se=TRUE: a matrix of estimated standard errors, of same dimension as coef.
- sample_size - Vector of sample sizes used for each phenotype

References

Examples
# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# convert to allele probabilities
aprobs <- genoprob_to_alleleprob(probs)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno[,1]
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
# calculate BLUPs of coefficients for chromosome 7
blup <- scan1blup(aprobs[,"7"], pheno, addcovar=covar)

# leave-one-chromosome-out kinship matrix for chr 7
kinship7 <- calc_kinship(probs, "loco")[["7"]]

# calculate BLUPs of coefficients for chromosome 7, adjusting for residual polygenic effect
blup_pg <- scan1blup(aprobs[,"7"], pheno, kinship7, addcovar=covar)

---

**scan1coef**

*Calculate QTL effects in scan along one chromosome*

**Description**

Calculate QTL effects in scan along one chromosome with a single-QTL model using Haley-Knott regression or a linear mixed model (the latter to account for a residual polygenic effect), with possible allowance for covariates.

**Usage**

```r
scan1coef(
    genoprobs,
    pheno,
    kinship = NULL,
    addcovar = NULL,
    nullcovar = NULL,
    intcovar = NULL,
    weights = NULL,
    contrasts = NULL,
    model = c("normal", "binary"),
    zerosum = TRUE,
    se = FALSE,
    hsq = NULL,
    reml = TRUE,
    ...
)
```

**Arguments**

- **genoprobs**: Genotype probabilities as calculated by `calc_genoprob()`.
- **pheno**: A numeric vector of phenotype values (just one phenotype, not a matrix of them).
- **kinship**: Optional kinship matrix, or a list of kinship matrices (one per chromosome), in order to use the LOCO (leave one chromosome out) method.
- **addcovar**: An optional numeric matrix of additive covariates.
nullcovar An optional numeric matrix of additional additive covariates that are used under the null hypothesis (of no QTL) but not under the alternative (with a QTL). This is needed for the X chromosome, where we might need sex as a additive covariate under the null hypothesis, but we wouldn’t want to include it under the alternative as it would be collinear with the QTL effects. Only used if kinship is provided but hsq is not, to get estimate of residual heritability.

intcovar An optional numeric matrix of interactive covariates.

weights An optional numeric vector of positive weights for the individuals. As with the other inputs, it must have names for individual identifiers.

contrasts An optional numeric matrix of genotype contrasts, size genotypes x genotypes. For an intercross, you might use cbind(mu=c(1,1,1), a=c(-1, 0, 1), d=c(0, 1, 0)) to get mean, additive effect, and dominance effect. The default is the identity matrix.

model Indicates whether to use a normal model (least squares) or binary model (logistic regression) for the phenotype. If model="binary", the phenotypes must have values in [0, 1].

zerosum If TRUE, force the genotype or allele coefficients sum to 0 by subtracting their mean and add another column with the mean. Ignored if contrasts is provided.

se If TRUE, also calculate the standard errors.

hsq (Optional) residual heritability; used only if kinship provided.

reml If kinship provided: if reml=TRUE, use REML; otherwise maximum likelihood.

... Additional control parameters; see Details;

Details

For each of the inputs, the row names are used as individual identifiers, to align individuals.

If kinship is absent, Haley-Knott regression is performed. If kinship is provided, a linear mixed model is used, with a polygenic effect estimated under the null hypothesis of no (major) QTL, and then taken as fixed as known in the genome scan.

If contrasts is provided, the genotype probability matrix, P, is post-multiplied by the contrasts matrix, A, prior to fitting the model. So we use P·A as the X matrix in the model. One might view the rows of A⁻¹ as the set of contrasts, as the estimated effects are the estimated genotype effects pre-multiplied by A⁻¹.

The ... argument can contain several additional control parameters; suspended for simplicity (or confusion, depending on your point of view). tol is used as a tolerance value for linear regression by QR decomposition (in determining whether columns are linearly dependent on others and should be omitted); default 1e-12. maxit is the maximum number of iterations for converence of the iterative algorithm used when model=binary. bintol is used as a tolerance for converence for the iterative algorithm used when model=binary. eta_max is the maximum value for the "linear predictor" in the case model="binary" (a bit of a technicality to avoid fitted values exactly at 0 or 1).

Value

An object of class "scan1coef": a matrix of estimated regression coefficients, of dimension positions x number of effects. The number of effects is n_genotypes + n_addcovar + (n_genotypes-1)*n_intcovar. May also contain the following attributes:
• **SE** - Present if se=TRUE: a matrix of estimated standard errors, of same dimension as coef.

• **sample_size** - Vector of sample sizes used for each phenotype

**References**


**Examples**

```r
# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno[,1]
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)

# calculate coefficients for chromosome 7
coef <- scan1coef(probs[,"7"], pheno, addcovar=covar)

# leave-one-chromosome-out kinship matrix for chr 7
kinship7 <- calc_kinship(probs, "loco")[["7"]]

# calculate coefficients for chromosome 7, adjusting for residual polygenic effect
coef_pg <- scan1coef(probs[,"7"], pheno, kinship7, addcovar=covar)
```

---

**scan1max**

*Maximum LOD score from genome scan with a single-QTL model*

**Description**

Maximum LOD score from genome scan with a single-QTL model by Haley-Knott regression or a linear mixed model, with possible allowance for covariates.
Usage

\[ \text{scan1max}( \]
  \text{genoprobs},
  \text{pheno},
  \text{kinship} = \text{NULL},
  \text{addcovar} = \text{NULL},
  \text{Xcovar} = \text{NULL},
  \text{intcovar} = \text{NULL},
  \text{weights} = \text{NULL},
  \text{reml} = \text{TRUE},
  \text{model} = \text{c("normal", "binary")},
  \text{hsq} = \text{NULL},
  \text{by_chr} = \text{FALSE},
  \text{cores} = 1,
  \ldots
\)

Arguments

- **genoprobs**: Genotype probabilities as calculated by `calc_genoprob()`. 
- **pheno**: A numeric matrix of phenotypes, individuals x phenotypes. 
- **kinship**: Optional kinship matrix, or a list of kinship matrices (one per chromosome), in order to use the LOCO (leave one chromosome out) method. 
- **addcovar**: An optional numeric matrix of additive covariates. 
- **Xcovar**: An optional numeric matrix with additional additive covariates used for null hypothesis when scanning the X chromosome. 
- **intcovar**: An numeric optional matrix of interactive covariates. 
- **weights**: An optional numeric vector of positive weights for the individuals. As with the other inputs, it must have names for individual identifiers. 
- **reml**: If kinship provided: if \text{reml=TRUE}, use REML; otherwise maximum likelihood. 
- **model**: Indicates whether to use a normal model (least squares) or binary model (logistic regression) for the phenotype. If \text{model="binary"}, the phenotypes must have values in \([0, 1]\). 
- **hsq**: Considered only if kinship is provided, in which case this is taken as the assumed value for the residual heritability. It should be a vector with length corresponding to the number of columns in pheno, or (if kinship corresponds to a list of LOCO kinship matrices) a matrix with dimension \text{length(kinship) x ncol(pheno)}. 
- **by_chr**: If TRUE, save the individual chromosome maxima. 
- **cores**: Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`. Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.
- \ldots: Additional control parameters; see Details.

Details

Equivalent to running `scan1()` and then saving the column maxima, with some savings in memory usage.
Value

Either a vector of genome-wide maximum LOD scores, or if by_chr is TRUE, a matrix with the chromosome-specific maxima, with the rows being the chromosomes and the columns being the phenotypes.

See Also

scan1(), scan1perm()

Examples

# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$map, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)

# perform genome scan
out <- scan1max(probs, pheno, addcovar=covar, Xcovar=Xcovar)

---

scan1perm

Permutation test for genome scan with a single-QTL model

Description

Permutation test for a genome scan with a single-QTL model by Haley-Knott regression or a linear mixed model, with possible allowance for covariates.

Usage

scan1perm(
  genopros,
  pheno,
  kinship = NULL,
  addcovar = NULL,
  Xcovar = NULL,
  intcovar = NULL,
weights = NULL,
reml = TRUE,
model = c("normal", "binary"),
n_perm = 1,
perm_Xsp = FALSE,
perm_strata = NULL,
chr_lengths = NULL,
cores = 1,
  ...
)

Arguments

Genotype probabilities as calculated by \texttt{calc\_genoprob()}.  
A numeric matrix of phenotypes, individuals x phenotypes.  
Optional kinship matrix, or a list of kinship matrices (one per chromosome), in order to use the LOCO (leave one chromosome out) method.  
An optional numeric matrix of additive covariates.  
An optional numeric matrix with additional additive covariates used for null hypothesis when scanning the X chromosome.  
An optional numeric matrix of interactive covariates.  
An optional numeric vector of positive weights for the individuals. As with the other inputs, it must have \texttt{names} for individual identifiers.  
If kinship provided: if \texttt{reml=TRUE}, use REML; otherwise maximum likelihood.  
Indicates whether to use a normal model (least squares) or binary model (logistic regression) for the phenotype. If \texttt{model="binary"}, the phenotypes must have values in $[0,1]$.  
Number of permutation replicates.  
If \texttt{TRUE}, do separate permutations for the autosomes and the X chromosome.  
Vector of strata, for a stratified permutation test. Should be named in the same way as the rows of \texttt{pheno}. The unique values define the strata.  
Lengths of the chromosomes; needed only if \texttt{perm\_Xsp=TRUE}. See \texttt{chr\_lengths()}.  
Number of CPU cores to use, for parallel calculations. (If 0, use \texttt{parallel::detectCores()}. Alternatively, this can be links to a set of cluster sockets, as produced by \texttt{parallel::makeCluster()}.  
Additional control parameters; see Details.

Details

If \texttt{kinship} is not provided, so that analysis proceeds by Haley-Knott regression, we permute the rows of the phenotype data; the same permutations are also applied to the rows of the covariates (\texttt{addcovar}, \texttt{Xcovar}, and \texttt{intcovar}) are permuted.

If \texttt{kinship} is provided, we instead permute the rows of the genotype data and fit an LMM with the same residual heritability (estimated under the null hypothesis of no QTL).
If `Xcovar` is provided and `perm_strata=NULL`, we do a stratified permutation test with the strata defined by the rows of `Xcovar`. If a simple permutation test is desired, provide `perm_strata` that is a vector containing a single repeated value.

The ... argument can contain several additional control parameters; suspended for simplicity (or confusion, depending on your point of view). `tol` is used as a tolerance value for linear regression by QR decomposition (in determining whether columns are linearly dependent on others and should be omitted); default `1e-12`. `maxit` is the maximum number of iterations for convergence of the iterative algorithm used when `model=binary`. `bintol` is used as a tolerance for convergence for the iterative algorithm used when `model=binary`. `eta_max` is the maximum value for the "linear predictor" in the case `model="binary"` (a bit of a technicality to avoid fitted values exactly at 0 or 1).

**Value**

If `perm_Xsp=FALSE`, the result is matrix of genome-wide maximum LOD scores, permutation replicates x phenotypes. If `perm_Xsp=TRUE`, the result is a list of two matrices, one for the autosomes and one for the X chromosome. The object is given class "`scan1perm`".

**References**


**See Also**

`scan1()`, `chr_lengths()`, `mat2strata()`

**Examples**

```r
# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
```
Xcovar <- get_x_covar(iron)

# strata for permutations
perm_strata <- mat2strata(Xcovar)

# permutations with genome scan (just 3 replicates, for illustration)
operm <- scan1perm(probs, pheno, addcovar=covar, Xcovar=Xcovar,
                    n_perm=3, perm_strata=perm_strata)
summary(operm)

# leave-one-chromosome-out kinship matrices
kinship <- calc_kinship(probs, "loco")

# permutations of genome scan with a linear mixed model
operm_lmm <- scan1perm(probs, pheno, kinship, covar, Xcovar, n_perm=3,
                       perm_Xsp=TRUE, perm_strata=perm_strata,
                       chr_lengths=chr_lengths(map))
summary(operm_lmm)

---

**scan1snps**  
*Single-QTL genome scan at imputed SNPs*

**Description**

Perform a single-QTL scan across the genome or a defined region at SNPs genotyped in the founders, by Haley-Knott regression or a linear mixed model, with possible allowance for covariates.

**Usage**

```r
scan1snps(
  genoprobs,  
  map,        
  pheno,      
  kinship = NULL,  
  addcovar = NULL,  
  Xcovar = NULL,  
  intcovar = NULL,  
  weights = NULL,  
  reml = TRUE,  
  model = c("normal", "binary"),  
  query_func = NULL,  
  chr = NULL,  
  start = NULL,  
  end = NULL,  
  snpinfo = NULL,  
  batch_length = 20,
```
keep_all_snps = FALSE,
cores = 1,
... )

Arguments

geno_probs Genotype probabilities as calculated by calc_genoprob().
map Physical map for the positions in the geno_probs object: A list of numeric vec-
tors; each vector gives marker positions for a single chromosome.
pheno A numeric matrix of phenotypes, individuals x phenotypes.
kinship Optional kinship matrix, or a list of kinship matrices (one per chromosome), in
order to use the LOCO (leave one chromosome out) method.
addcovar An optional numeric matrix of additive covariates.
Xcovar An optional numeric matrix with additional additive covariates used for null
hypothesis when scanning the X chromosome.
intcovar An optional numeric matrix of interactive covariates.
weights An optional numeric vector of positive weights for the individuals. As with the
other inputs, it must have names for individual identifiers.
reml If kinship provided: if reml=TRUE, use REML; otherwise maximum likelihood.
model Indicates whether to use a normal model (least squares) or binary model (logistic
regression) for the phenotype. If model="binary", the phenotypes must have
values in [0,1].
query_func Function for querying SNP information; see create_variant_query_func()).
  Takes arguments chr, start, end, (with start and end in the units in map,
generally Mbp), and returns a data frame containing the columns snp, chr, pos,
  and sdp. (See snpinfo below.)
chr Chromosome or chromosomes to scan
start Position defining the start of an interval to scan. Should be a single number, and
  if provided, chr should also have length 1.
end Position defining the end of an interval to scan. Should be a single number, and
  if provided, chr should also have length 1.
snpinfo Optional data frame of SNPs to scan; if provided, query_func, chr, start, and
  end are ignored. Should contain the following columns:
  • chr - Character string or factor with chromosome
  • pos - Position (in same units as in the "map").
  • sdp - Strain distribution pattern: an integer, between 1 and $2^n - 2$ where
    $n$ is the number of strains, whose binary encoding indicates the founder
genotypes
  • snp - Character string with SNP identifier (if missing, the rownames are
    used).
batch_length Interval length (in units of map, generally Mbp) to scan at one time.
SNPs are grouped into equivalence classes based on position and founder genotypes; if `keep_all_snps=FALSE`, the return value will contain information only on the indexed SNPs (one per equivalence class).

Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`.) Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.

Additional control parameters passed to `scan1()`.

The analysis proceeds as follows:

- Call `query_func()` to grab all SNPs over a region.
- Use `index_snps()` to group SNPs into equivalence classes.
- Use `genoprob_to_snpprob()` to convert genoprobs to SNP probabilities.
- Use `scan1()` to do a single-QTL scan at the SNPs.

A list with two components: `lod` (matrix of LOD scores) and `snpinfo` (a data frame of SNPs that were scanned, including columns `index` which indicates groups of equivalent SNPs).

### See Also

`scan1()`, `genoprob_to_snpprob()`, `index_snps()`, `create_variant_query_func()`, `plot_snpasso()`

### Examples

```r
## Not run:
# load example data and calculate genotype probabilities
file <- paste0("https://raw.githubusercontent.com/rqtl/",
               "qtl2data/main/DOex/DOex.zip")
DOex <- read_cross2(file)
probs <- calc_genoprob(DOex, error_prob=0.002)

snpdb_file <- system.file("extdata", "cc_variants_small.sqlite", package="qtl2")
queryf <- create_variant_query_func(snpdb_file)

out <- scan1snps(probs, DOex$pmap, DOex$pheno, query_func=queryf, chr=2, start=97, end=98)
## End(Not run)
```
**sdp2char**  
*Convert strain distribution patterns to character strings*

**Description**  
Convert a vector of numeric codes for strain distribution patterns to character strings.

**Usage**  
```
sdp2char(sdp, n_strains = NULL, strains = NULL)
```

**Arguments**  
- **sdp**: Vector of strain distribution patterns (integers between 1 and \(2^n - 2\) where \(n\) is the number of strains.
- **n_strains**: Number of founder strains (if missing but `strains` is provided, we use the length of `strains`)
- **strains**: Vector of single-letter codes for the strains

**Value**  
Vector of character strings with the two groups of alleles separated by a vertical bar (`|`).

**See Also**  
`invert_sdp()`, `calc_sdp()`

**Examples**  
```
sdp <- c(m1=1, m2=12, m3=240)
sdp2char(sdp, 8)
sdp2char(sdp, strains=c("A", "B", "1", "D", "Z", "C", "P", "W"))
```

---

**sim_geno**  
*Simulate genotypes given observed marker data*

**Description**  
Uses a hidden Markov model to simulate from the joint distribution \(Pr(g | O)\) where \(g\) is the underlying sequence of true genotypes and \(O\) is the observed multipoint marker data, with possible allowance for genotyping errors.
**Usage**

```r
sim_geneno(
    cross,
    map = NULL,
    n_draws = 1,
    error_prob = 0.0001,
    map_function = c("haldane", "kosambi", "c-f", "morgan"),
    lowmem = FALSE,
    quiet = TRUE,
    cores = 1
)
```

**Arguments**

- `cross` Object of class "cross2". For details, see the R/qtl2 developer guide.
- `map` Genetic map of markers. May include pseudomarker locations (that is, locations that are not within the marker genotype data). If NULL, the genetic map in `cross` is used.
- `n_draws` Number of simulations to perform.
- `error_prob` Assumed genotyping error probability
- `map_function` Character string indicating the map function to use to convert genetic distances to recombination fractions.
- `lowmem` If FALSE, split individuals into groups with common sex and crossinfo and then precalculate the transition matrices for a chromosome; potentially a lot faster but using more memory.
- `quiet` If FALSE, print progress messages.
- `cores` Number of CPU cores to use, for parallel calculations. (If 0, use `parallel::detectCores()`. Alternatively, this can be links to a set of cluster sockets, as produced by `parallel::makeCluster()`.

**Details**

After performing the backward equations, we draw from \( Pr(g_1 = v | O) \) and then \( Pr(g_{k+1} = v | O, g_k = u) \).

**Value**

An object of class "sim_geneno": a list of three-dimensional arrays of imputed genotypes, individuals x positions x draws. Also contains three attributes:

- `crosstype` - The cross type of the input cross.
- `is_x_chr` - Logical vector indicating whether chromosomes are to be treated as the X chromosome or not, from input cross.
- `alleles` - Vector of allele codes, from input cross.

**See Also**

`cbind.sim_geneno()`, `rbind.sim_geneno()`
smooth_gmap

Examples

grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
map_w_pmar <- insert_pseudomarkers(grav2$gmap, step=1)
draws <- sim_geno(grav2, map_w_pmar, n_draws=4, error_prob=0.002)

____________________________
smooth_gmap Smooth genetic map

Description

Smooth a genetic map by mixing it with a bit of constant recombination (using a separate recombination rate for each chromosome), to eliminate intervals that have exactly 0 recombination.

Usage

smooth_gmap(gmap, pmap, alpha = 0.02)

Arguments

gmap Genetic map, as a list of numeric vectors; each vector gives marker positions for a single chromosome.
pmap Physical map, as a list of numeric vectors; each vector gives marker positions for a single chromosome, with the same chromosomes and markers as gmap.
alpa Proportion of mixture to take from constant recombination.

Details

An interval of genetic length \(d_g\) and physical length \(d_p\) is changed to have length \((1 - \alpha)d_g + \alpha d_p r\) where \(r = L_g / L_p\) is the chromosome-specific recombination rate.

Value

A genetic map like the input gmap, but smoothed by mixing it with a proportion alpha of constant recombination on each chromosome.

See Also

unsmooth_gmap()

Examples

iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
gmap_adj <- smooth_gmap(iron$gmap, iron$pmap)
Description

Pull out a specified set of individuals and/or chromosomes from the results of `calc_genoprob()`.

Usage

```r
## S3 method for class 'calc_genoprob'
subset(x, ind = NULL, chr = NULL, ...)

## S3 method for class 'calc_genoprob'
x[ind = NULL, chr = NULL]
```

Arguments

- `x` Genotype probabilities as output from `calc_genoprob()`.
- `ind` A vector of individuals: numeric indices, logical values, or character string IDs.
- `chr` A vector of chromosomes: logical values, or character string IDs. Numbers are interpreted as character string IDs.
- `...` Ignored.

Value

An object of class "calc_genoprob", like the input, with the selected individuals and/or chromosomes; see `calc_genoprob()`.

Examples

```r
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))

pr <- calc_genoprob(grav2)
# keep just individuals 1:5, chromosome 2
prsub <- pr[1:5,2]
# keep just chromosome 2
prsub2 <- pr[,2]
```
Description

Pull out a specified set of individuals and/or chromosomes from a cross2 object.

Usage

```r
## S3 method for class 'cross2'
subset(x, ind = NULL, chr = NULL, ...)
## S3 method for class 'cross2'
x[ind = NULL, chr = NULL]
```

Arguments

- `x`: An object of class "cross2", as output by `read_cross2()`. For details, see the R/qtl2 developer guide.
- `ind`: A vector of individuals: numeric indices, logical values, or character string IDs.
- `chr`: A vector of chromosomes: numeric indices, logical values, or character string IDs.
- `...`: Ignored.

Details

When subsetting by individual, if `ind` is numeric, they’re assumed to be numeric indices; if character strings, they’re assumed to be individual IDs. `ind` can be numeric or logical only if the genotype, phenotype, and covariate data all have the same individuals in the same order.

When subsetting by chromosome, `chr` is always converted to character strings and treated as chromosome IDs. So if there are three chromosomes with IDs "18", "19", and "X", `mycross[,18]` will give the first of the chromosomes (labeled "18") and `mycross[,3]` will give an error.

When using character string IDs for `ind` or `chr`, you can use "negative" subscripts to indicate exclusions, for example `mycross[,c("-18", "-X")]) or mycross["-Mouse2501",]`. But you can’t mix "positive" and "negative" subscripts, and if any of the individuals has an ID that begins with "-", you can’t use negative subscripts like this.

Value

The input cross2 object, with the selected individuals and/or chromosomes.

Warning

The order of the two arguments is reversed relative to the related function in R/qtl.
Examples

grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
# keep individuals 1-20 and chromosomes 3 and 4
grav2sub <- grav2[1:20, c(3,4)]
# keep just chromosome 1
grav2_c1 <- grav2[,1]

subset.geno

Subsetting imputed genotypes

Description

Pull out a specified set of individuals and/or chromosomes from the results of sim.geno().

Usage

## S3 method for class 'sim.geno'
subset(x, ind = NULL, chr = NULL, ...)
## S3 method for class 'sim.geno'
x[ind = NULL, chr = NULL]

Arguments

x

Imputed genotypes as output from sim.geno().

ind

A vector of individuals: numeric indices, logical values, or character string IDs

chr

A vector of chromosomes: logical values, or character string IDs. Numbers are interpreted as character string IDs.

...

Ignored.

Value

An object of class "sim_geno", like the input with the selected individuals and/or chromosomes; see sim.geno().

Examples

grav2 <- read.cross2(system.file("extdata", "grav2.zip", package="qtl2"))

dr <- sim.geno(grav2, n.draws=4)
# keep just individuals 1:5, chromosome 2
drsub <- dr[1:5,2]
# keep just chromosome 2
drsub2 <- dr[,2]
subset.viterbi

Subsetting Viterbi results

Description

Pull out a specified set of individuals and/or chromosomes from the results of viterbi()

Usage

```r
## S3 method for class 'viterbi'
subset(x, ind = NULL, chr = NULL, ...)

## S3 method for class 'viterbi'
x[ind = NULL, chr = NULL]
```

Arguments

- `x` Imputed genotypes as output from viterbi().
- `ind` A vector of individuals: numeric indices, logical values, or character string IDs
- `chr` A vector of chromosomes: logical values, or character string IDs. Numbers are interpreted as character string IDs.
- `...` Ignored.

Value

An object of class "viterbi", like the input, with the selected individuals and/or chromosomes; see viterbi().

Examples

```r
grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))

g <- viterbi(grav2)
# keep just individuals 1:5, chromosome 2
gsub <- g[1:5,2]
# keep just chromosome 2
gsub2 <- g[,2]
```
Description

Subset the output of scan1() by chromosome or column

Usage

subset_scan1(x, map = NULL, chr = NULL, lodcolumn = NULL, ...)

## S3 method for class 'scan1'
subset(x, map = NULL, chr = NULL, lodcolumn = NULL, ...)

Arguments

- **x**: An object of class "scan1" as returned by scan1().
- **map**: A list of vectors of marker positions, as produced by insert_pseudomarkers().
- **chr**: Vector of chromosomes.
- **lodcolumn**: Vector of integers or character strings indicating the LOD score columns, either as a numeric indexes or column names.
- **...**: Ignored

Value

Object of class "scan1", like the input, but subset by chromosome and/or column. See scan1().

Examples

```r
# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)

# perform genome scan
out <- scan1(probs, pheno, addcovar=covar, Xcovar=Xcovar)

# pull out chromosome 8
```
out_c8 <- subset(out, map, chr="8")

# just the second column on chromosome 2
out_c2_spleen <- subset(out, map, "2", "spleen")

# all positions, but just the "liver" column
out_spleen <- subset(out, map, lodcolumn="spleen")

---

### summary.cross2

**Summary of cross2 object**

**Description**

Summarize a cross2 object

**Usage**

```r
## S3 method for class 'cross2'
summary(object, ...)
```

**Arguments**

- `object` An object of class "cross2", as output by `read_cross2()`. For details, see the R/qtl2 developer guide.
- `...` Ignored.

**Value**

None.

**See Also**

- `basic_summaries`

---

### summary_compare_geneno

**Basic summary of compare_geneno object**

**Description**

From results of `compare_geneno()`, show pairs of individuals with similar genotypes.
**summary_scan1perm**

**Summarize scan1perm results**

**Description**

Summarize permutation test results from `scan1perm()`, as significance thresholds.

**Usage**

```r
summary_scan1perm(object, alpha = 0.05)
```

```r
## S3 method for class 'scan1perm'
summary(object, alpha = 0.05, ...)
```
Arguments

object  An object of class "scanoneperm", as output by scan1perm()
alpha  Vector of significance levels
...  Ignored

Details
In the case of X-chromosome-specific permutations (when scan1perm() was run with perm_Xsp=TRUE, we follow the approach of Broman et al. (2006) to get separate thresholds for the autosomes and X chromosome, using

Let \( L_A \) and \( L_X \) be total the genetic lengths of the autosomes and X chromosome, respectively, and let \( L_T = L_A + L_X \). Then in place of \( \alpha \), we use

\[
\alpha_A = 1 - (1 - \alpha) \frac{L_A}{L_T}
\]

as the significance level for the autosomes and

\[
\alpha_X = 1 - (1 - \alpha) \frac{L_X}{L_T}
\]

as the significance level for the X chromosome.

Value
An object of class summary.scan1perm. If scan1perm() was run with perm_Xsp=FALSE, this is a single matrix of significance thresholds, with rows being significance levels and columns being the columns in the input. If scan1perm() was run with perm_Xsp=TRUE, this is a list of two matrices, with the significance thresholds for the autosomes and X chromosome, respectively.

The result has an attribute "n_perm" that has the numbers of permutation replicates (either a matrix or a list of two matrices).

References

Examples
# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)

# permutations with genome scan (just 3 replicates, for illustration)
operm <- scan1perm(probs, pheno, addcovar=covar, Xcovar=Xcovar,
                    n_perm=3)
summary(operm, alpha=c(0.20, 0.05))

---

top_snps

Create table of top snp associations

Description
Create a table of the top snp associations

Usage
top_snps(
  scan1_output,
  snpinfo,
  lodcolumn = 1,
  chr = NULL,
  drop = 1.5,
  show_all_snps = TRUE
)

Arguments

scan1_output  Output of scan1(). Should contain a component "snpinfo", as when scan1() is run with SNP probabilities produced by genoprob_to_snpprob().

snpinfo        Data frame with SNP information with the following columns (the last three are generally derived with index_snps()):
  • chr - Character string or factor with chromosome
  • pos - Position (in same units as in the "map" attribute in genoprops.
  • sdp - Strain distribution pattern: an integer, between 1 and $2^n - 2$ where $n$ is the number of strains, whose binary encoding indicates the founder genotypes
  • snp - Character string with SNP identifier (if missing, the rownames are used).
  • index - Indices that indicate equivalent groups of SNPs, calculated by index_snps()
  • intervals - Indexes that indicate which marker intervals the SNPs reside.
  • on_map - Indicate whether SNP coincides with a marker in the genoprops
lodcolumn  
Selected LOD score column to (a numeric index, or a character string for a column name). Only one value allowed.

chr  
Selected chromosome; only one value allowed.

drop  
Show all SNPs with LOD score within this amount of the maximum SNP association.

show_all_snps  
If TRUE, expand to show all SNPs.

Value  
Data frame like the input snpinfo with just the selected subset of rows, and with an added column with the LOD score.

See Also  

index_snps(), genoprob_to_snpprob(), scan1snps(), plot_snpasso()

Examples  

```r
## Not run:
# load example DO data from web
file <- paste0("https://raw.githubusercontent.com/rqtl/",
    "qtl2data/main/DOex/DOex.zip")
DOex <- read_cross2(file)
# subset to chr 2
DOex <- DOex[, "2"]
# calculate genotype probabilities and convert to allele probabilities
pr <- calc_genoprob(DOex, error_prob=0.002)
apr <- genoprob_to_alleleprob(pr)
# query function for grabbing info about variants in region
dbfile <- system.file("extdata", "cc_variants_small.sqlite", package="qtl2")
query_variants <- create_variant_query_func(dbfile)
# SNP association scan, keep information on all SNPs
out_snps <- scan1snps(apr, DOex$pmap, DOex$pheno, query_func=query_variants,
    chr=2, start=97, end=98, keep_all_snps=TRUE)
# table with top SNPs
top_snps(out_snps$lod, out_snps$snpinfo)
# top SNPs among the distinct subset at which calculations were performed
top_snps(out_snps$lod, out_snps$snpinfo, show_all_snps=FALSE)
# top SNPs within 0.5 LOD of max
top_snps(out_snps$lod, out_snps$snpinfo, drop=0.5)
## End(Not run)
```
unsmooth_gmap

Unsmooth genetic map

Description

Performs the reverse operation of smooth_gmap(), in case one wants to go back to the original genetic map.

Usage

unsmooth_gmap(gmap, pmap, alpha = 0.02)

Arguments

gmap Genetic map, as a list of numeric vectors; each vector gives marker positions for a single chromosome.
pmap Physical map, as a list of numeric vectors; each vector gives marker positions for a single chromosome, with the same chromosomes and markers as gmap.
alpha Proportion of mixture to take from constant recombination.

Details

An interval of genetic length $d_g$ and physical length $d_p$ is changed to have length $(d_g - \alpha d_p r)/(1 - \alpha)$ where $r = L_g/L_p$ is the chromosome-specific recombination rate.

Value

A genetic map like the input gmap, but with the reverse operation of smooth_gmap() applied, provided that exactly the same physical map and alpha are used.

See Also

smooth_gmap()

Examples

iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
gmap_adj <- smooth_gmap(iron$gmap, iron$pmap)
gmap_back <- unsmooth_gmap(gmap_adj, iron$pmap)
viterbi

Calculate most probable sequence of genotypes

Description

Uses a hidden Markov model to calculate arg max Pr(g | O) where g is the underlying sequence of true genotypes and O is the observed multipoint marker data, with possible allowance for genotyping errors.

Usage

viterbi(
  cross,  
  map = NULL,  
  error_prob = 0.0001,  
  map_function = c("haldane", "kosambi", "c-f", "morgan"),  
  lowmem = FALSE,  
  quiet = TRUE,  
  cores = 1
)

Arguments

cross Object of class "cross2". For details, see the R/qtl2 developer guide.

map Genetic map of markers. May include pseudomarker locations (that is, locations that are not within the marker genotype data). If NULL, the genetic map in cross is used.

error_prob Assumed genotyping error probability

map_function Character string indicating the map function to use to convert genetic distances to recombination fractions.

lowmem If FALSE, split individuals into groups with common sex and crossinfo and then precalculate the transition matrices for a chromosome; potentially a lot faster but using more memory.

quiet If FALSE, print progress messages.

cores Number of CPU cores to use, for parallel calculations. (If 0, use parallel::detectCores(). Alternatively, this can be links to a set of cluster sockets, as produced by parallel::makeCluster().

Details

We use a hidden Markov model to find, for each individual on each chromosome, the most probable sequence of underlying genotypes given the observed marker data. Note that we break ties at random, and our method for doing this may introduce some bias.

Consider the results with caution; the most probable sequence can have very low probability, and can have features that are quite unusual (for example, the number of recombination events can be too small). In most cases, the results of a single imputation with sim_geno() will be more realistic.
Value

An object of class "viterbi": a list of two-dimensional arrays of imputed genotypes, individuals x positions. Also contains three attributes:

- crosstype - The cross type of the input cross.
- is_x_chr - Logical vector indicating whether chromosomes are to be treated as the X chromosome or not, from input cross.
- alleles - Vector of allele codes, from input cross.

See Also

sim_geno(), maxmarg(), cbind.viterbi(), rbind.viterbi()

Examples

grav2 <- read_cross2(system.file("extdata", "grav2.zip", package="qtl2"))
map_w_pmar <- insert_pseudomarkers(grav2$gmap, step=1)
g <- viterbi(grav2, map_w_pmar, error_prob=0.002)

Description

Write the control file (in YAML or JSON) needed by read_cross2() for a set of QTL data.

Usage

write_control_file(
  output_file,
  crosstype = NULL,
  geno_file = NULL,
  founder_geno_file = NULL,
  gmap_file = NULL,
  pmap_file = NULL,
  pheno_file = NULL,
  covar_file = NULL,
  phenocovar_file = NULL,
  sex_file = NULL,
  sex_covar = NULL,
  sex_codes = NULL,
  crossinfo_file = NULL,
  crossinfo_covar = NULL,
  crossinfo_codes = NULL,
  geno_codes = NULL,
  alleles = NULL,
  xchr = NULL,
)
```r
sep = "",
na.strings = c("-", "NA"),
comment.char = "#",
geno_transposed = FALSE,
founder_geno_transposed = FALSE,
pheno_transposed = FALSE,
covar_transposed = FALSE,
phenocovar_transposed = FALSE,
description = NULL,
comments = NULL,
overwrite = FALSE
)
```

**Arguments**

- `output_file`: File name (with path) of the YAML or JSON file to be created, as a character string. If extension is `.json`, JSON format is used; otherwise, YAML is used.
- `crosstype`: Character string with the cross type.
- `geno_file`: File name for genotype data.
- `founder_geno_file`: File name for the founder genotype data.
- `gmap_file`: File name for genetic map.
- `pmap_file`: File name for the physical map.
- `pheno_file`: File name for the phenotype data.
- `covar_file`: File name for the covariate data.
- `phenocovar_file`: File name for the phenotype covariate data (i.e., metadata about the phenotypes).
- `sex_file`: File name for the individuals’ sex. (Specify just one of `sex_file` or `sex_covar`.)
- `sex_covar`: Column name in the covariate data that corresponds to sex. (Specify just one of `sex_file` or `sex_covar`.)
- `sex_codes`: Named vector of character strings specifying the encoding of sex. The names attribute should be the codes used in the data files; the values within the vector should be "female" and "male".
- `crossinfo_file`: File name for the cross_info data. (Specify just one of `crossinfo_file` or `crossinfo_covar`.)
- `crossinfo_covar`: Column name in the covariate data that corresponds to the cross_info data. (Specify just one of `crossinfo_file` or `crossinfo_covar`.)
- `crossinfo_codes`: In the case that there is a single cross info column (whether in a file or as a covariate), you can provide a named vector of character strings specifying the encoding of cross_info. The names attribute should be the codes used; the values within the vector should be the codes to which they will be converted (for example, 0 and 1 for an intercross).
write_control_file  

```r
write_control_file(geno_codes, alleles, xchr, sep, na.strings, comment.char, geno_transposed = FALSE, founder_geno_transposed = FALSE, pheno_transposed = FALSE, covar_transposed = FALSE, phenocovar_transposed = FALSE, description, comments, overwrite = FALSE)
```

**Arguments**

- `geno_codes`: Named vector specifying the encoding of genotypes. The names attribute has the codes used within the genotype and founder genotype data files; the values within the vector should be the integers to which the genotypes will be converted.
- `alleles`: Vector of single-character codes for the founder alleles.
- `xchr`: Character string with the ID for the X chromosome.
- `sep`: Character string that separates columns in the data files.
- `na.strings`: Vector of character strings with codes to be treated as missing values.
- `comment.char`: Character string that is used as initial character in a set of leading comment lines in the data files.
- `geno_transposed`: If TRUE, genotype file is transposed (with markers as rows).
- `founder_geno_transposed`: If TRUE, founder genotype file is transposed (with markers as rows).
- `pheno_transposed`: If TRUE, phenotype file is transposed (with phenotypes as rows).
- `covar_transposed`: If TRUE, covariate file is transposed (with covariates as rows).
- `phenocovar_transposed`: If TRUE, phenotype covariate file is transposed (with phenotype covariates as rows).
- `description`: Optional character string describing the data.
- `comments`: Vector of character strings to be inserted as comments at the top of the file (in the case of YAML), with each string as a line. For JSON, the comments are instead included within the control object.
- `overwrite`: If TRUE, overwrite file if it exists. If FALSE (the default) and the file exists, stop with an error.

**Details**

This function takes a set of parameters and creates the control file (in YAML or JSON format) needed for the new input data file format for R/qtl2. See the sample data files and the vignette describing the input file format.

**Value**

(Invisibly) The data structure that was written.

**See Also**

read_cross2(), sample data files at [https://kbroman.org/qtl2/pages/sampledata.html](https://kbroman.org/qtl2/pages/sampledata.html)
Examples

# Control file for the sample dataset, grav2
grav2_control_file <- file.path(tempdir(), "grav2.yaml")
write_control_file(grav2_control_file,
   crosstype="riself",
   geno_file="grav2_geno.csv",
   gmap_file="grav2_gmap.csv",
   pheno_file="grav2_pheno.csv",
   phenocovar_file="grav2_phenocovar.csv",
   geno_codes=c(L=1L, C=2L),
   alleles=c("L", "C"),
   na.strings=c("-", "NA"))

# Control file for the sample dataset, iron
iron_control_file <- file.path(tempdir(), "iron.yaml")
write_control_file(iron_control_file,
   crosstype="f2",
   geno_file="iron_geno.csv",
   gmap_file="iron_gmap.csv",
   pheno_file="iron_pheno.csv",
   covar_file="iron_covar.csv",
   phenocovar_file="iron_phenocovar.csv",
   geno_codes=c(SS=1L, SB=2L, BB=3L),
   sex_covar="sex",
   sex_codes=c(f="female", m="male"),
   crossinfo_covar="cross_direction",
   crossinfo_codes=c("(SxB)x(SxB)"=0L, "(BxS)x(BxS)"=1L),
   xchr="X",
   alleles=c("S", "B"),
   na.strings=c("-", "NA"))

# Remove these files, to clean up temporary directory
unlink(c(grav2_control_file, iron_control_file))

---

xpos_scan1

Get x-axis position for genomic location

Description

For a plot of scan1() results, get the x-axis location that corresponds to a particular genomic location (chromosome ID and position).

Usage

xpos_scan1(map, chr = NULL, gap = NULL, thechr, thepos)

Arguments

map        A list of vectors of marker positions, as produced by insert_pseudomarkers().
Selected chromosomes that were plotted (if used in the call to `plot_scan1()`).

The gap between chromosomes used in the call to `plot_scan1()`.

Vector of chromosome IDs

Vector of chromosomal positions

**Details**

The `thechr` and `thepos` should be the same length, or should have length 1 (in which case they are expanded to the length of the other vector).

**Value**

A vector of x-axis locations.

**Examples**

```r
# read data
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))

# insert pseudomarkers into map
map <- insert_pseudomarkers(iron$gmap, step=1)

# calculate genotype probabilities
probs <- calc_genoprob(iron, map, error_prob=0.002)

# grab phenotypes and covariates; ensure that covariates have names attribute
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)

# perform genome scan
out <- scan1(probs, pheno, addcovar=covar, Xcovar=Xcovar)

# plot the results for selected chromosomes
ylim <- c(0, maxlod(out)*1.02) # need to strip class to get overall max LOD
chr <- c(2,7,8,9,15,16)
plot(out, map, chr=chr, ylim=ylim)
plot(out, map, lodcolumn=2, chr=chr, col="violetred", add=TRUE)
legend("topleft", lwd=2, col=c("darkslateblue", "violetred"), colnames(out),
bg="gray90")

# Use xpos_scan1 to add points at the peaks
# first find the peaks with LOD > 3
peaks <- find_peaks(out, map)

# keep just the peaks for chromosomes that were plotted
peaks <- peaks[peaks$chr %in% chr,]

# find x-axis positions
xpos <- xpos_scan1(map, chr=chr, thechr=peaks$chr, thepos=peaks$pos)
```
# point colors
ptcolor <- c("darkslateblue", "violetred")[match(peaks$lodcolumn, c("liver", "spleen"))]

# plot points
points(xpos, peaks$lod, pch=21, bg=ptcolor)

zip_datafiles

Zip a set of data files

Description
Zip a set of data files (in format read by read_cross2()).

Usage
zip_datafiles(control_file, zip_file = NULL, overwrite = FALSE, quiet = TRUE)

Arguments
control_file Character string with path to the control file (YAML or JSON) containing all of
the control information.
zip_file Name of zip file to use. If NULL, we use the stem of control_file but with a
.zip extension.
overwrite If TRUE, overwrite file if it exists. If FALSE (the default) and the file exists, stop
with an error.
quiet If FALSE, print progress messages.

Details
The input control_file is the control file (in YAML or JSON format) to be read by read_cross2().
(See the sample data files and the vignette describing the input file format.)
The utils::zip() function is used to do the zipping.
The files should all be contained within the directory where the control_file sits, or in a subdi-
rectory of that directory. If file paths use . . . , these get stripped by zip, and so the resulting zip file
may not work with read_cross2().

Value
Character string with the file name of the zip file that was created.

See Also
read_cross2(), sample data files at https://kbroman.org/qtl2/pages/sampledata.html
Examples

```r
## Not run:
zipfile <- file.path(tempdir(), "grav2.zip")
zip_datafiles("grav2.yaml", zipfile)

## End(Not run)
```
Index

* **IO**
  - read_cross2, 110
  - read_pheno, 113
  - zip_datafiles, 154

* **datasets**
  - CCcolors, 27

* **graphics**
  - plot_compare_geno, 81

* **hgraphics**
  - plot_genes, 81
  - plot_sdp, 95

* **htest**
  - chisq_colpairs, 28

* **utilities**
  - basic_summaries, 5
  - calc_entropy, 10
  - calc_errorlod, 11
  - calc_genoprob, 12
  - calc_grid, 15
  - calc_kinship, 16
  - calc_raw_founder_maf, 18
  - calc_raw_geno_freq, 19
  - calc_raw_het, 20
  - calc_raw_maf, 20
  - compare_geno, 33
  - convert2cross2, 36
  - est_map, 46
  - genoprob_to_alleleprob, 58
  - get_x_covar, 62
  - guess_phase, 62
  - insert_pseudomarkers, 65
  - map_to_grid, 71
  - max_compare_geno, 75
  - predict_snpgeno, 99
  - print.cross2, 100
  - probs_to_grid, 101
  - recode_snps, 114
  - scale_kinship, 118
  - sim_geno, 134
  - smooth_gmap, 136
  - subset.calc_genoprob, 137
  - subset.cross2, 138
  - subset.sim_geno, 139
  - subset.viterbi, 140
  - summary.cross2, 142
  - summary_compare_geno, 142
  - unsmooth_gmap, 147
  - write_control_file, 149

  .calc_genoprob (subset.calc_genoprob), 137
  .cross2 (subset.cross2), 138
  .sim_geno (subset.sim_geno), 139
  .viterbi (subset.viterbi), 140

  add_threshold, 4

  base::cbind(), 26
  base::paste(), 72
  base::print(), 100
  basic_summaries, 5, 142
  batch.cols, 7
  batch.cols(), 8
  batch.vec, 8
  batch.vec(), 8
  bayes_int, 9
  bayes_int(), 54, 71

  c.scan1perm (rbind.scan1perm), 107
  calc_entropy, 10
  calc_errorlod, 11
  calc_errorlod(), 99
  calc_genofreq, 14
  calc_genoprob, 12
  calc_grid, 15
  calc_grid(), 66, 72, 102
INDEX

157

calc_het, 16
calc_kinship, 16
calc_kinship(), 118, 121
calc_raw_founder_maf, 18
calc_raw_founder_maf(), 20, 21, 114
calc_raw_geno_freq, 19
calc_raw_geno_freq(), 20, 21
calc_raw_het, 20
calc_raw_het(), 19, 21
calc_raw_maf, 20
calc_raw_maf(), 18–20, 114
calc_sdp, 21
calc_sdp(), 68, 96, 134
cbind.calc_genoprob, 22
cbind.calc_genoprob(), 105
cbind.scan1, 23
cbind.scan1(), 106, 121
cbind.scan1perm, 24
cbind.scan1perm(), 107
cbind.sim_geno, 25
cbind.sim_geno(), 108, 135
cbind.viterbi, 26
cbind.viterbi(), 109, 149
cbind_expand, 26
CCaltcolors (CCcolors), 27
CCcolors, 27, 80
CCorigcolors (CCcolors), 27
check_cross2, 28
chisq_colpairs, 28
chr_lengths, 29
chr_lengths(), 129, 130
chr_names (basic_summaries), 5
clean, 30
clean.calc_genoprob (clean_genoprob), 30
clean.calc_genoprob(), 30
clean.scan1 (clean_scan1), 32
clean.scan1(), 30
clean_genoprob, 30
clean_scan1, 32
compare_geno, 33
compare_geno(), 75, 76, 81, 142, 143
compare_genoprob, 34
compare_maps, 35
convert2cross2, 36
count_xo, 37
count_xo(), 69
covar_names (basic_summaries), 5
create_gene_query_func, 38
create.snpinfo, 39
create_variant_query_func, 40
create_variant_query_func(), 132, 133
data.table::fread(), 111, 112
decomp_kinship, 42
drop_markers, 43
drop_markers(), 44, 104
drop_nullmarkers, 43
drop_nullmarkers(), 43, 104
est_herit, 44
est_map, 46
find_ibd_segments, 47
find_index_snp, 49
find_index_snp(), 51, 64
find_map_gaps, 50
find_map_gaps(), 115
find_marker, 51
find_marker(), 49, 52, 57, 103
find_markerpos, 52
find_markerpos(), 51
find_peaks, 53
find_peaks(), 10, 71, 77, 87, 90, 91
fit1, 55
fit1(), 103
founders (basic_summaries), 5
genoprob_to_alleleprob, 58
genoprob_to_alleleprob(), 13, 17, 34, 83, 84, 86
genoprob_to_snpprob, 59
genoprob_to_snpprob(), 40, 64, 97, 133, 145, 146
get_common_ids, 61
get_x_c covar, 62
get_x_c covar(), 72
graphics::image(), 84, 86
graphics::segments(), 5
guess_phase, 62
guess_phase(), 89
hist(), 81
ind_ids (basic_summaries), 5
ind_ids_c covar (basic_summaries), 5
ind_ids_geno (basic_summaries), 5
ind_ids_gnp (basic_summaries), 5
ind_ids_pheno (basic_summaries), 5
index_snps(), 40, 49, 53, 59, 60, 76, 97, 133, 145, 146
insert_pseudomarkers, 65
insert_pseudomarkers(), 9, 13, 15, 53, 64, 67, 70, 72, 73, 76, 79, 94, 141, 152
interp_genoprob, 66
interp_map, 67
invert_sdp, 68
invert_sdp(), 21, 96, 134
locate_xo, 69
locate_xo(), 37
lod_int, 70
lod_int(), 10, 54
map_to_grid, 71
map_to_grid(), 15, 102
marker_names (basic_summaries), 5
mat2strata, 72
mat2strata(), 130
max.compare_genos (max.compare_genos), 75
max.scan1 (max.scan1), 76
max_compare_genos, 75
max_scan1, 76
maxlod, 73
maxmarg, 74
maxmarg(), 34, 37, 63, 69, 89, 92, 99, 149
n_chr (basic_summaries), 5
n_covar (basic_summaries), 5
n_founders (basic_summaries), 5
n_ind (basic_summaries), 5
n_ind_cvar (basic_summaries), 5
n_ind_geno (basic_summaries), 5
n_ind_gnp (basic_summaries), 5
n_ind_pheno (basic_summaries), 5
n_mar (basic_summaries), 5
n_missing, 77
n_pheno (basic_summaries), 5
n_phenocovar (basic_summaries), 5
n_typed (n_missing), 77
phenocovar_names (basic_summaries), 5
plot.calc_genoprob (plot_genoprob), 83
plot.compare_genos (plot_compare_genos), 81
plot.scan (plot_scan), 94
plot.scanl (plot_scanl), 94
plot_coef, 78
plot_coef(), 93, 94, 98
plot_coefCC (plot_coefC), 78
plot_coefCC(), 94, 98
plot.compare_genos, 81
plot_genes, 81
plot_genoprob, 83
plot_genoprob(), 86
plot_genoprobcomp, 85
plot_genoprobcomp(), 35, 84
plot_lodpeaks, 87
plot_lodpeaks(), 91
plot_onegeno, 88
plot_peaks, 90
plot_peaks(), 87
plot_pxg, 92
plot_scan1, 94
plot_scan1(), 80, 98, 153
plot_sdp, 95
plot_snpasso, 96
plot_snpasso(), 80, 94, 133, 146
predict_snpqgeno, 99
print.cross2, 100
print.summary.compare_genos (summary_compare_genos), 142
print.summary.scan1perm, 100
probs_to_grid, 101
probs_to_grid(), 15, 72
pull_genoprobint, 102
pull_genoprobint(), 51, 103
pull_genoprobpos, 103
pull_genoprobpos(), 51, 57, 103
pull_markers, 104
pull_markers(), 43, 44
qt2version, 105
qt::read.cross(), 36
rbind.calc_genoprob, 105
rbind.calc_genoprob(), 22
INDEX

rbind.scan1, 106
rbind.scan1(), 23, 121
rbind.scan1perm, 107
rbind.scan1perm(), 24
rbind.sim_gen0, 108
rbind.sim_gen0(), 25, 135
rbind.viterbi, 109
rbind.viterbi(), 26, 149
read_cross2, 110
read_cross2(), 6, 28, 37, 59, 78, 100, 113, 138, 142, 149, 151, 154
read_csv, 111
read_csv(), 112
read_csv_numer, 112
read_csv_numer(), 111
read_pheno, 113
read_pheno(), 110
recode_snps, 114
recode_snps(), 18–21
reduce_map_gaps, 115
reduce_map_gaps(), 50
reduce_markers, 115
replace_ids, 117
replace_ids(), 81
scale_kinship, 118
scan1, 119
scan1(), 9, 10, 23, 24, 32, 53, 54, 70, 71, 73, 76, 94, 97, 106, 107, 128, 130, 133, 141, 145, 152
scan1blup, 121
scan1blup(), 56
scan1coef, 124
scan1coef(), 79, 121
scan1max, 126
scan1max(), 121
scan1perm, 128
scan1perm(), 24, 29, 72, 107, 121, 128, 143, 144
scan1snps, 131
scan1snps(), 39, 40, 53, 60, 64, 76, 146
sdp2char, 134
sdp2char(), 21, 68
sim_gen0, 134
sim_gen0(), 25, 37, 75, 108, 118, 139, 148, 149
smooth_gmap, 136
smooth_gmap(), 147
subset.calc_genoprob, 137
subset.cross2, 138
subset.scan1(subset_scan1), 141
subset.sim_gen0, 139
subset.viterbi, 140
subset_scan1, 141
summary.compare_gen0
(summary_compare_gen0), 142
summary.compare_gen0(), 143
summary.cross2, 142
summary.cross2(), 7
summary.scan1perm(summary_scan1perm), 143
summary_compare_gen0, 142
summary_scan1perm, 143
summary_scan1perm(), 100, 101
top_snps, 145
tot_mar (basic_summaries), 5
unsMOOTH_gmap, 147
unsMOOTH_gmap(), 136
utils::zip(), 154
viterbi, 148
viterbi(), 26, 37, 69, 75, 99, 109, 118, 140
write_control_file, 149
write_control_file(), 110
xpos_scan1, 152
zip_datafiles, 154