Package 'mpMap2'

April 23, 2020

Type Package

Title Genetic Analysis of Multi-Parent Recombinant Inbred Lines

Version 1.0.3

Date 2020-04-23

Author Rohan Shah [aut, cre],
Emma Huang [aut],
Matthew Morell [ctb],
Alex Whan [ctb],
Colin Cavanagh [ctb]

Maintainer Rohan Shah <cran@bookshah.com>

Description Constructing linkage maps, reconstructing haplotypes, estimating linkage disequilibrium and quantitative trait loci (QTL) mapping in multi-parent Recombinant Inbred Lines designs.

License GPL-2

SystemRequirements C++11

LazyLoad yes

Depends R (>= 3.5.0)

LinkingTo Rcpp

Suggests testthat, knitr, rmarkdown, gridExtra, Heatplus

Imports ggplot2, Matrix, methods, qtl, igraph, fastcluster, pryr, nnls, RColorBrewer, jsonlite, progress, stats, sn, car

RoxygenNote 7.0.2

Collate 'Pillai.R' 'Rcpp_exceptions.R' 'map-class.R' 'addExtraMarkerFromRawCall.R' 'addExtraMarkers.R' 'canSkipValidity.R' 'pedigree-class.R' 'hetData-class.R' 'geneticData-class.R' 'lg-class.R' 'rawSymmetricMatrix.R' 'rf-class.R' 'mpcross-class.R' 'additionOperators.R' 'as.mpInterval.R' 'assignFounderPattern.R' 'assignFounderPatternPrototype.R' 'backcrossPedigree.R' 'biparentalDominant.R' 'callFromMap.R' 'changeMarkerPosition.R' 'combineGenotypes.R' 'combineKeepRF.R'

'compressedProbabilities.R' 'computeAllEpistaticChiSquared.R'
'computeGenotypeProbabilities.R' 'createSNPTemplate.R'
'detailedPedigree-class.R' 'eightWayPedigreeImproperFunnels.R'
'eightWayPedigreeRandomFunnels.R'
'eightWayPedigreeSingleFunnel.R' 'estimateMap.R'
'estimateMapFromImputation.R' 'estimateRF.R'
'estimateRFSingleDesign.R' 'expand.R' 'expandedProbabilities.R'
'exportMapToPretzl.R' 'extraImputationPoints.R' 'f2Pedigree.R'
'finals.R' 'fixedNumberOfFounderAlleles.R' 'formGroups.R'
'founders.R' 'fourParentPedigreeRandomFunnels.R'
'fourParentPedigreeSingleFunnel.R' 'fullHetData.R'
'generateGridPositions.R' 'generateIntervalMidPoints.R'
'getAllFunnels.R' 'getChromosomes.R'
'getIntercrossingAndSelfingGenerations.R' 'getPositions.R'
'hetData.R' 'identC.R' 'imputationGenerics.R' 'impute.R'
'imputeFounders.R' 'jitterMap.R' 'lineNames.R'
'listCodingErrors.R' 'mapFunctions.R' 'markers.R' 'mpcross.R'
'multiparentSNP.R' 'multiparentSNPPrototype.R' 'nFounders.R'
'nLines.R' 'nMarkers.R' 'num_threads.R' 'orderCross.R'
'pedigree.R' 'pedigreeGraph-class.R' 'pedigreeGraph.R'
'pedigreeToGraph.R' 'plot.R' 'plotMosaic.R'
'plotProbabilities.R' 'print.R' 'probabilityData.R'
'purdyToPedigree.R' 'redact.R' 'removeHets.R'
'reorderPedigree.R' 'reverseChromosomes.R' 'rilPedigree.R'
'roxygen.R' 'selfing.R' 'show.R' 'simulateMPCross.R'
'simulatePhenotypes.R' 'singleLocusProbabilities.R'
'sixteenParentPedigreeRandomFunnels.R' 'stripPedigree.R'
'subset.R' 'testDistortion.R' 'toMpMap.R'
'transposeProbabilities.R' 'twoParentPedigree.R' 'validation.R'

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-04-23 16:10:14 UTC

R topics documented:

`+,mpcrossMapped,mpcrossMapped-method`
Combine mpcross objects

Description

Combine two mpcross objects into a single object

Usage

```
## S4 method for signature 'mpcrossMapped,mpcrossMapped'
e1 + e2

## S4 method for signature 'mpcross,mpcross'
e1 + e2

## S4 method for signature 'mpcrossRF,mpcrossRF'
e1 + e2

## S4 method for signature 'mpcrossRF,mpcross'
e1 + e2
```

Arguments

`e1`	An mpcross object
`e2`	Another mpcross object

Details

These addition operators combine multiple objects of classes `mpcross` or `mpcrossMapped` into a single object. The input objects may contain recombination fraction data, or may have associated genetic maps. The operators try to keep whatever extra data is in the input objects, and will warn if data is discarded. Data will be discarded if, for example, one of the objects contains recombination fraction data and the other does not.

In general, the combined object will contain the input objects as separate experiments. In special cases, the datasets may actually be combined as a single experiment. For example, if the input objects contains disjoint sets of markers, but the same genetic lines, then the datasets will be combined. Similarly, if the input objects contain the same genetic markers, but disjoint sets of genetic lines, then the datasets will be combined.

Internally this function redirects to another generic named `addMpMap2`, because this generic allows for optional named arguments.

Value

A combined object that contains the data from both `e1` and `e2`.

`addExtraMarkerFromRawCall` *Add an extra marker from raw calling data*

Description

Add an extra marker to a map, based on raw calling data, using a QTL-mapping style approach.

Usage

```
addExtraMarkerFromRawCall(
  mpcrossMapped,
  newMarker,
  useOnlyExtraImputationPoints = TRUE
)
```

Arguments

`mpcrossMapped`	An object of class `mpcrossMapped` (dataset with a map), which must include imputed IBD genotypes and recombination fraction data.
`newMarker`	A matrix containing the raw data for the marker to add.
`useOnlyExtraImputationPoints`	Should we only attempt to add the new marker at points at which imputation data has been generated, which are *not* markers?

Details

This function uses a QTL-mapping style approach to test for where an extra marker should be added to an existing map. The code uses the imputation data at a collection of points, and the *raw calling data* for the extra marker. The raw calling data must be bivariate.

Test statistics measuring the association of the new marker to a point are computed using a multivariate analysis of variance approach. If the imputed genotype at a point is independent of the data for the new marker, then the new marker probably should *not* be mapped to that point. If the imputed genotype at a point and the data for the new marker are strongly *dependent*, then the new marker *should* probably be mapped to that point. Dependence and independence are measured using an F-test.

By default the set of points at which the new marker is considered for addition is the set of points at which imputation data is available, *which are not markers*. The intention is that this set of points should be an equally spaced grid of points; this reduces the number of tests that are performed, as generally there are far fewer points in the grid, than there are markers. After the new marker is added, local reordering will need to be performed anyway, making any loss in accuracy by using the grid of points largely irrelevant. Setting `useOnlyExtraImputationPoints` to `FALSE` means that every marker position will also be used as a possible position for the new marker (this is not recommended).

Value

An object of class `addExtraMarkersStatistics` containing the test statistic values and the genetic map used to generate them.

`addExtraMarkers` *Add extra markers*

Description

Add extra markers to a map, using a QTL-mapping style approach.

Usage

```
addExtraMarkers(
  mpcrossMapped,
  newMarkers,
  useOnlyExtraImputationPoints = TRUE,
  reorderRadius = 103,
  maxOffset = 50,
  knownChromosome,
  imputationArgs = NULL,
  onlyStatistics = FALSE,
  orderCrossArgs = list(),
  verbose = TRUE,
  reorder = TRUE
)
```

Arguments

`mpcrossMapped`	An object of class `mpcrossMapped` (dataset with a map), which must include imputed IBD genotypes and recombination fraction data.
`newMarkers`	An object of class `mpcross` containing the new markers to add.
`useOnlyExtraImputationPoints`	Should we only attempt to add the new marker at points at which imputation data has been generated, which are *not* markers? Currently this must be `TRUE`. In future `FALSE` may be allowed.
`reorderRadius`	The width of the region (in terms of number of markers) in which to attempt to reorder, after the extra markers are added.
`maxOffset`	The maxOffset parameter for the call to `estimateMap`, which is used to re-estimate the map (locally), after the additional markers are added.
`knownChromosome`	The name of a chromosome, if the extra markers are known to go on a specific chromosome
`imputationArgs`	A list containing additional arguments to `imputeFounders`.

onlyStatistics	If this argument is TRUE, then only the chi-squared test statistic values are computed, and the markers are not added.
orderCrossArgs	A list containing additional arguments to orderCross.
verbose	Should extra logging output be generated?
reorder	Should local reordering be performed after the extra markers are added?

Details

This function uses a QTL-mapping style approach to add extra markers to an existing map. The code uses the imputation data at a collection of points, and the marker alleles for the *first* marker of the extra markers. If the imputed genotype at a point is *independent* from the genotype at the new marker, then the new marker probably should *not* be mapped to that point. If the imputed genotype at a point and the marker allele are *strongly dependent*, then the new marker *should* probably be mapped to that point. Dependence and independence are measured using a chi-squared test stastistic for independence. *All the extra markers* are then mapped to the position where the test statistic is largest. It is recommended that only single markers be added at a time, unless you are extremely confident that all the extra markers should be located at the same position.

Currently the set of points at which the new markers are considered for addition is the set of points at which imputation data is available, *which are not markers*. The intention is that this set of points should be an equally spaced grid of points; this reduced the number of tests that are performed, as generally there are far fewer points in the grid, than there are markers. After the new marker is added, local reordering will need to be performed anyway, making any loss in accuracy by using the grid of points largely irrelevant. In future it may be possible to use the set of all marker positions as the set of points at which tests are performed, by setting useOnlyExtraImputationPoints to FALSE.

Once the extra markers have added, local reordering is optionally performed, depending on argument reordering. The radius of the region on which reordering is performed, in terms of the number of markers, is reorderRadius.

Once the optional reordering step has been performed, the map is recomputed locally, to account for the addition of the extra markers. The argument maxOffset is passed through to estimateMap. Finally, the imputation data will be recomputed if imputationArgs is not NULL; in that case, imputationArgs should contain a list of arguments to imputeFounders. It is recommended that the imputation data be recomputed if further markers are to be added.

In some cases the user will want to apply a threshold to the maximum value of the test statistics, and only add the marker if the test statistics exceed the threshold. In this case the function should be called twice. For the first call, onlyStatistics should be set to FALSE. If the resulting test statistics exceed the threshold, then addExtrMarkers should be called again with onlyStatistics set to TRUE.

Value

If onlyStatistics was set to TRUE, an object of class addExtraMarkersStatistics containing the test statistic values. If onlyStatistics was set to FALSE, a list containing the test statistic values in entry statistics and in entry object, a new object containing the input object with the new markers added.

Examples

```
data(simulatedFourParentData)
#Create object that includes the correct map
mapped <- new("mpcrossMapped", simulatedFourParentData, map = simulatedFourParentMap)
#Remove marker number 50. Normally the map is discarded, but we specify to keep it.
removedMiddle <- subset(mapped, markers = (1:101)[-50], keepMap = TRUE)
#Compute imputation data, at all the markers, and an equally spaced grid of points
removedMiddle <- imputeFounders(removedMiddle, errorProb = 0.02,
extraPositions = generateGridPositions(1))
#Estimate recombination fractions
removedMiddle <- estimateRF(removedMiddle)
#Get out the extra marker to add
extraMarker <- subset(simulatedFourParentData, markers = 50)
#Add the extra marker, without doing any local reordering. After the marker is added,
# recompute the imputation data, using the same arguments as previously.
withExtra <- addExtraMarkers(mpcrossMapped = removedMiddle, newMarkers = extraMarker,
reorder = FALSE, imputationArgs = list(errorProb = 0.02,
extraPositions = generateGridPositions(1)))$object
```

---

as.mpInterval *Convert mpcross object to MPWGAIM format*

Description

Convert an object of class mpcrossMapped to the format used by MPWGAIM.

Usage

```
as.mpInterval(
  object,
  type = "mpMarker",
  positions,
  homozygoteMissingProb,
  heterozygoteMissingProb,
  errorProb
)
```

Arguments

object	The object of class mpcrossMapped to convert
type	The type of MPWGAIM object to output. Must be "mpMarker" or "mpInterval"
positions	In the case of mpMarker format, the positions at which the IBD probabilities should be output. Must be either "all" (all positions for which IBD probabilities are available) or "marker" (only marker positions).
homozygoteMissingProb	Used as an input to computeGenotypeProbabilitiesInternal, if the IBD probabilities need to be calculated.

`heterozygoteMissingProb`
: Used as an input to `computeGenotypeProbabilitiesInternal`, if the IBD probabilities need to be calculated.

`errorProb`
: Used as an input to `computeGenotypeProbabilitiesInternal`, if the IBD probabilities need to be calculated.

Details

MPWGAIM is a package for performing QTL analysis using multi-parent populations. This function outputs a data object suitable for input to MPWGAIM. The output object can be in MPWGAIMs `mpMarker` or `mpInterval` formats. See the documentation of MPWGAIM for further information.

Value

An object of class `mpMarker` or `mpInterval`, which are formats specified by package mpwgaim.

`assignFounderPattern` *Set founder genotypes*

Description

Set founder genotypes

Usage

```
assignFounderPattern(founderMatrix)
```

Arguments

`founderMatrix` The new matrix of founder genotypes

Details

Set the founder genotypes to a specified matrix, for an object with fully informative markers. This can allow the same set of founder genotypes to be used for multiple simulation runs.

Value

An object of internal class `assignFounderPattern`, suitable for application to an object of class `mpcross` using the addition operation.

backcrossPedigree *Generate a backcross pedigree which starts from inbred founders*

Description

Generate a backcross pedigree which starts from inbred founders

Usage

```
backcrossPedigree(populationSize)
```

Arguments

populationSize	The size of the generated population.

Details

Generate a backcross pedigree which starts from inbred founders

Value

An object of class `detailedPedigree` representing the experimental design, suitable for simulation using simulateMPCross.

Examples

```
pedigree <- backcrossPedigree(1000)
#This pedigree is automatically marked as involving finite generations of selfing.
selfing(pedigree)
```

biparentalDominant *Make markers in a biparental cross dominant*

Description

Change the markers in a biparental cross from fully informative to dominant. The dominant founder is chosen randomly for every marker. The transformation is applied to an object using the addition operator, see the example below for details.

Usage

```
biparentalDominant()
```

Value

An object of internal type `biparentalDominant`, which can be combined with an object of class `mpcross` using the addition operator.

Examples

```
#Simulate an F2 design
f2Pedigree <- f2Pedigree(1000)
map <- qtl::sim.map(len = 100, n.mar = 11, include.x=FALSE)
cross <- simulateMPCross(map = map, pedigree = f2Pedigree, mapFunction = haldane, seed = 1)
founders(cross)
finals(cross)[1:10,]
#The heterozygotes are initially coded as 3
hetData(cross)[[1]]
#Make all markers dominant
dominantCross <- cross + biparentalDominant()
founders(dominantCross)
finals(dominantCross)[1:10,]
#The heterozygotes are now coded the same as one of the homozygotes
hetData(dominantCross)[1:4]
```

---

callFromMap *Call markers based on an existing map*

Description

This function uses an existing genetic map to call genetic markers, including markers polymorphic on multiple chromosomes.

Usage

```
callFromMap(
  rawData,
  thresholdChromosomes = 100,
  thresholdAlleleClusters = c(1e-10, 1e-20, 1e-30, 1e-40),
  maxChromosomes = 2,
  existingImputations,
  tDistributionPValue = 0.6,
  useOnlyExtraImputationPoints = TRUE,
  ...
)
```

Arguments

`rawData`	Raw data for a genetic marker.
`thresholdChromosomes`	The test-statistic threshold for declaring a marker to be polymorphic on a chromosome.
`thresholdAlleleClusters`	The p-value threshold for declaring two underlying founder alleles to have different marker alleles. Multiple possible values should be input.
`maxChromosomes`	The maximum number of chromosomes that a marker can be polymorphic on

`existingImputations`
: An object of class mpcrossMapped from the mpMap2 package, containing data about imputed underlying genotypes.

`tDistributionPValue`
: Paramater controlling the size of each detected cluster, ranging from 0 to 1. Small values result in small clusters, and large values result in large clusters.

`useOnlyExtraImputationPoints`
: Should we only use the non-marker positions to identify the correct locations?

`...`
: Extra arguments. Only `existingLocalisationStatistics` is supported, mostly so the example can run quickly.

Details

This function uses an existing genetic map to call a genetic marker. There are a number of advantages to this approach

1. It can correctly call markers which are polymorphic on multiple chromosomes, therefore converting one marker into two.
2. It avoids incorrectly calling markers polymorphic on multiple chromosomes. Incorrect calling can lead to supurious genetic interactions.
3. It can call markers that initially appear to be monomorphic in the population.
4. It can call additional marker alleles for markers that would otherwise be ignored.

Once a genetic map has been constructed, it should be used to impute underlying founder genotypes at an equally spaced grid of points using function `imputeFounders`. The steps in the algorithm are as follows:

1. Determine which chromosomes the marker is associated to, and where on those chromosomes. This is determined using function `addExtraMarkerFromRawCall`, which is itself based on a manova model. The marker is assumed associated to chromosomes for which the test statistic is greater than `thresholdChromosomes`. An appropriate value for `thresholdChromosomes` can be determined by looking at the results of `addExtraMarkerFromRawCall`, for a number of different markers.
2. Determine the distribution of marker alleles, at all the associated genetic locations. This is done by taking the founders to be the vertices of a graph, and connecting founders which seem to part of the same marker allele. The resulting graph should be a union of disjoint complete graphs (cliques).
3. We now have a preliminary assignment of marker alleles to lines, where the assignment may be of 1, 2, 3 or more *different* marker alleles, depending on how many chromosomes the marker is associated with. For example, if the marker is associated with two chromosomes, then there will be two marker alleles for each line. For each unique combination of marker alleles, we take the lines which have that assignment of marker alleles, and fit a skew-t distribution.
4. For each fitted distribution, determine a confidence region using p-value `tDistributionPValue`.
5. Use these confidence regions to construct marker calls at each associated location.

Value

At the minimum, a list containing an entry `called` indicating whether the marker could be successfully called. If it could, other entries are returned.

overallAssignment Defines clusters within the data.

classificationsPerPosition Defines genotype calls per genetic location to which the marker was mapped.

clusterBoundaries Contours giving the boundaries of each cluster in `overallAssignment`.

preliminaryGroups The preliminary groups based on IBD imputations, which the final genotype calls are built from.

pValuesMatrices The matrices of p-values used to form a graph, and therefore identify founder alleles.

Examples

```
data(eightParentSubsetMap)
data(wsnp_Ku_rep_c103074_89904851)
data(callFromMapExampleLocalisationStatistics)
library(ggplot2)
library(gridExtra)
#We use an existing set of localisation statistics, to make the example faster
called <- callFromMap(rawData = as.matrix(wsnp_Ku_rep_c103074_89904851), existingImputations =
    eightParentSubsetMap, useOnlyExtraImputationPoints = TRUE, tDistributionPValue = 0.8,
   thresholdChromosomes = 80, existingLocalisationStatistics = existingLocalisationStatistics)
plotData <- wsnp_Ku_rep_c103074_89904851
plotData$genotype1B <- factor(called$classificationsPerPosition$Chr1BLoc31$finals)
plotData$imputed1B <- factor(imputationData(eightParentSubsetMap)[, "Chr1BLoc31"])
plotData$genotype1D <- factor(called$classificationsPerPosition$Chr1DLoc16$finals)
plotData$imputed1D <- factor(imputationData(eightParentSubsetMap)[, "Chr1DLoc16"])

plotImputations1B <- ggplot(plotData, mapping = aes(x = theta, y = r, color = imputed1B)) +
    geom_point() + theme_bw() + ggtitle("Imputed genotype, 1B") +
    guides(color=guide_legend(title="IBD genotype"))

called1B <- ggplot(plotData, mapping = aes(x = theta, y = r, color = genotype1B)) +
    geom_point() + theme_bw() + ggtitle("Called genotype, 1B") +
    guides(color=guide_legend(title="Called cluster")) + scale_color_manual(values =
    c("black", RColorBrewer::brewer.pal(n = 4, name = "Set1")))

plotImputations1D <- ggplot(plotData, mapping = aes(x = theta, y = r, color = imputed1D)) +
    geom_point() + theme_bw() + ggtitle("Imputed genotype, 1D") +
    guides(color=guide_legend(title="IBD genotype"))

called1D <- ggplot(plotData, mapping = aes(x = theta, y = r, color = genotype1D)) +
    geom_point() + theme_bw() + ggtitle("Called genotype, 1D") +
    guides(color=guide_legend(title="Called cluster")) +
   scale_color_manual(values = c("black",RColorBrewer::brewer.pal(n=3,name = "Set1")[1:2]))

grid.arrange(plotImputations1B, plotImputations1D, called1B, called1D)
```

changeMarkerPosition *Change the position of a single marker*

Description

Change the position of a single marker

Usage

```
changeMarkerPosition(mpcrossMapped, marker, newChromosome, newPosition)
```

Arguments

mpcrossMapped	The object of class mpcrossMapped, containing a marker to be modified
marker	The name of the marker to change
newChromosome	The new chromosome for the specified marker
newPosition	The new position for the specified marker in cM, on the new chromosome

Details

For an object of class mpcrossMapped, change the position of a single marker

Value

A copy of the input object, with the specified marker shifted to the new position and chromosome.

clusterOrderCross *Group markers into blocks and arrange those blocks*

Description

Group markers into blocks and arrange those blocks

Usage

```
clusterOrderCross(
  mpcrossLG,
  cool = 0.5,
  tmin = 0.1,
  nReps = 1,
  maxMove = 0,
  effortMultiplier = 1,
  randomStart = TRUE,
  nGroups
)
```

Arguments

mpcrossLG	An object of class mpcrossLG, containing genetic data and linkage groups.
cool	Rate of cooling
tmin	Minimum temperature
nReps	Number of independent replications of the simulated annealing algorithm
maxMove	Maximum number of positions by which to shift a single marker, as part of the simulated annealing. A value of zero indicates no limit.
effortMultiplier	Multiplier for the amount of computational effort
randomStart	If TRUE, start from the current ordering
nGroups	The number of groups to form using hierarchical clustering

Details

In some cases the number of markers is too large to reorder all markers on a chromosome. However, the problem becomes more tractable if the markers are already in a roughly correct ordering to start with. This function is intended to generate that roughly accurate ordering, and then subsequenty local reordering using orderCross can be applied to generate a final marker ordering.

The rough ordering is generated by forming some number of groups of markers, using hierarchical clustering. A consensus disimilarity between every group of markers is formed, and this is used to order the groups. That is, we decide whether the markers will be ordered as group 1, group 2, group 3, etc, or group 2, group 1, group 3, etc. The ordering of the markers within each group is unchanged.

Value

An object of class mpcrossLG, identical to the input except with the markers rearranged.

---

combineKeepRF *Combine mpcross objects, keeping recombination fraction data*

Description

Combine mpcross objects, keeping recombination fraction data

Usage

```
combineKeepRF(
  object1,
  object2,
  verbose = TRUE,
  gbLimit = -1,
  callEstimateRF = TRUE,
  skipValidity = FALSE
)
```

Arguments

object1	An object of class mpcrossRF
object2	Another object of class mpcrossRF
verbose	Passed straight through to estimateRF
gbLimit	Passed straight through to estimateRF
callEstimateRF	Should estimateRF be called, to compute any missing estimates?
skipValidity	Should we skip the validity check for object construction, in this function? Running the validity checks can be expensive, and in theory internal package code is trusted to generate valid objects.

Details

This function takes two objects containing disjoint sets of markers, each containing estimated recombination fractions for their individual sets of markers. A new object is returned that contains the combined set of markers, and also contains recombination fraction data.

This function is more efficient than other ways of achieving this, as it keeps the recombination fraction data contained in the original objects. If callEstimateRF is TRUE, it also computes the missing recombination fraction estimates between markers in different objects, using a call to estimateRF.

Value

A new object of class mpcrossRF containing the combined information of the two input objects.

---

computeAllEpistaticChiSquared

Compute chi-squared test statistics for independence

Description

Compute chi-squared test statistics for independence

Usage

```
computeAllEpistaticChiSquared(mpcrossMapped, verbose = TRUE)
```

Arguments

mpcrossMapped	An object of class mpcrossMapped with IBD probability data.
verbose	If this is TRUE a progress bar is generated

Details

This function computes what are (approximately) chi-squared test statistics for independence of the genotypes at different points on the genome. This computation is done using the IBD probability data. Significant non-independence between IBD probabilities at distant points on the same chromosome, or points on different chromosomes, can indicate non-standard genetic inheritance or selective pressure.

Value

A square matrix with rows and columns corresponding to genetic locations, and values corresponding to test statistics.

`computeGenotypeProbabilities` *Compute IBD genotype probabilities*

Description

Compute IBD genotype probabilities

Usage

```
computeGenotypeProbabilities(
  mpcrossMapped,
  homozygoteMissingProb = 1,
  heterozygoteMissingProb = 1,
  errorProb = 0,
  extraPositions = list()
)
```

Arguments

`mpcrossMapped`	An object of class `mpcrossMapped`, for which to compute the IBD genotype probabilities.
`homozygoteMissingProb`	The "probability" that a marker genotype that is truly homozygous will be marked as missing.
`heterozygoteMissingProb`	The "probability" that a marker genotype that is truly heterozygous will be marked as missing.
`errorProb`	The probability that a marker genotype is incorrect.
`extraPositions`	The extra positions at which to compute the IBD genotype probabilities. May be either a list with named components corresponding to chromosomes (simialr to a map) or a function which will be applied to the input object to generate the extra positions.

Details

This function computes the IBD genotype probabilities using a Hidden Marker Model (HMM) and the forward-backward algorithm. The HMM model is only an approximation to the underlying genetics, but it is a very good one.

There are a number of parameters to this model. `homozygoteMissingProb` gives the "probability" that a marker homozygote will be marked a missing. `heterozygoteMissingProb` gives the "probability" that a marker heterozygote will be marked as missing. We say "probability" because really

the important thing is the difference these two parameters, not the values themselves. If they are equal then a missing marker genotype contains no information. If codeheterozygoteMissingProb is relatively larger than `homozygoteMissingProb`, then missing marker genotypes suggests that the underlying genotype is heterozygous, provided enough missing marker values occur sequentially.

The key reason for introducing these paramters is that if `heterozygoteMissingProb` is relatively larger, then a dataset with no observed marker heterozygotes can still be used to estimate positions of underlying heterozygous genotypes, provided that heterozygous genotypes lead to consecutive missing marker genotype values.

The `errorProb` parameter gives the probability that a marker genotype is actually incorrect. In this case, it is assumed that the correct value for this marker genotype is random and uniformly distributed. This is different from assuming that the underlying genotype itself is random. If `errorProb` is zero, then it is not possible to have co-located markers with inconsistent genotypes, and if this occurs an error will be generated. `jitterMap` can be used to avoid this, but setting `errorProb` to some non-zero value is a much better solution.

It is also possible to generate IBD probabilities at non-marker positions. These extra positions are specified by the `extraPositions` options, which can be specified two ways. The first is by specifying a list with name entries, where the names correspond to chromosomes. Each named entry should be a named vector, with names corresponding to the names of the positions, and values corresponding to the positions in cM on that chromosome.

The second possibility is to specify a function, which will be applied to the input object of class `mpcrossMapped` to generate the extra positions. Two helper options are provided for this - `generateGridPositions` and `link{generateIntervalMidPoints}`.

Value

An object of class `mpcrossMapped` containing all information in the input object, and also estimated IDB probabilities.

`detailedPedigree-class`

Pedigree for simulation

Description

Class detailedPedigree is similar to the S4 class pedigree, except it also contains information about which lines are going to observed. This allows simulation of a data set with the given pedigree.

Usage

```
detailedPedigree(lineNames, mother, father, initial, observed, selfing)
```

Arguments

`lineNames`	The names assigned to the lines.
`mother`	The female parent of this line, given by name or by index within `lineNames`.

father	The male parent of this line, given by name or by index within lineNames.
initial	The founder lines, given by name or by index within lineNames.
observed	The lines which are observed in the final population, given by name or by index within lineNames.
selfing	Value determining whether or not subsequent analysis of populations generated from this pedigree should assume infinite generations of selfing. Possible values are "finite" and "infinite".

Value

An object of class detailedPedigree, suitable for simulation.

Functions

- detailedPedigree: Construct object of class detailedPedigree

Slots

initial The indices of the inbred founder lines in the pedigree. These founders lines must be the first lines in the pedigree.

observed A logical vector with one value per line in the pedigree. A value of TRUE indicates that this line will be genotyped.

See Also

pedigree-class, simulateMPCross, detailedPedigree

detailedPedigree-class

Examples

```
lineNames <- paste0("L", 1:10)
mother <- c(0, 0, 1, rep(3, 7))
father <- c(0, 0, 2, rep(2, 7))
initial <- 1:2
lineNames <- paste0("L", 1:10)
observed <- c(rep(FALSE, 3), rep(TRUE, 7))
detailedPedigreeObj <- detailedPedigree(mother = mother, father = father, initial = initial,
observed = observed, lineNames = lineNames, selfing = "finite")
```

---

eightParentPedigreeImproperFunnels

Generate an eight-parent pedigree with improper funnels

Description

Generate a eight-parent pedigree starting from inbred founders, where the founders in the funnels are not necessarily distinct.

Usage

```
eightParentPedigreeImproperFunnels(
  initialPopulationSize,
  selfingGenerations,
  nSeeds
)
```

Arguments

initialPopulationSize	The number of initially generated lines, whose genetic material is a mosaic of the eight founding lines. These lines are generated using three generations of structured mating.
selfingGenerations	The number of selfing generations at the end of the pedigree.
nSeeds	The number of progeny taken from each intercrossing line, or from each initially generated line (if no intercrossing is specified). These lines are then selfed according to selfingGenerations.

Details

Generate a eight-parent pedigree starting from inbred founders. The founders in the funnel for every line are chosen *with replacement*. So for any line from the final population, it is likely that some founding lines are absent from the corresponding funnel, and some appear multiple times.

Value

An object of class `detailedPedigree` representing the experimental design, suitable for simulation using simulateMPCross.

See Also

eightParentPedigreeSingleFunnel, fourParentPedigreeSingleFunnel, fourParentPedigreeRandomFunnels, twoParentPedigree

Examples

```
pedigree <- eightParentPedigreeImproperFunnels(initialPopulationSize = 10,
selfingGenerations = 0, nSeeds = 1)
#Generate map
map <- qtl::sim.map()
#Simulate data
cross <- simulateMPCross(map = map, pedigree = pedigree, mapFunction = haldane)
#Get out a list of funnels, which are rows of this matrix. Note that, of the values 1:8,
#    some are missing within a row, and some are repeated.
getAllFunnels(cross)
#convert the pedigree to a graph
pedigreeAsGraph <- pedigreeToGraph(pedigree)
#Plot it
plot(pedigreeAsGraph)
```

```
#Write it to a file in DOT format
```

`eightParentPedigreeRandomFunnels` *Generate an eight-parent pedigree, using random funnels*

Description

Generate a eight-parent pedigree starting from inbred founders, using a random funnel.

Usage

```
eightParentPedigreeRandomFunnels(
  initialPopulationSize,
  selfingGenerations,
  nSeeds = 1L,
  intercrossingGenerations
)
```

Arguments

`initialPopulationSize`	The number of initially generated lines, whose genetic material is a mosaic of the eight founding lines. These lines are generated using three generations of structured mating.
`selfingGenerations`	The number of selfing generations at the end of the pedigree.
`nSeeds`	The number of progeny taken from each intercrossing line, or from each initially generated line (if no intercrossing is specified). These lines are then selfed according to selfingGenerations.
`intercrossingGenerations`	The number of generations of random mating performed from the F1 generation. Population size is maintained at that specified by initialPopulationSize.

Value

An object of class `detailedPedigree` representing the experimental design, suitable for simulation using simulateMPCross.

See Also

`eightParentPedigreeSingleFunnel`, `fourParentPedigreeSingleFunnel`, `fourParentPedigreeRandomFunnels`, `twoParentPedigree`

Examples

```
pedigree <- eightParentPedigreeRandomFunnels(initialPopulationSize = 10,
selfingGenerations = 0, nSeeds = 1, intercrossingGenerations = 10)
#Generate map
map <- qtl::sim.map()
#Simulate data
cross <- simulateMPCross(map = map, pedigree = pedigree, mapFunction = haldane)
#Get out a list of funnels, which are rows of this matrix. For this pedigree, every
#    funnel is a random ordering of 1:8.
getAllFunnels(cross)
#convert the pedigree to a graph
pedigreeAsGraph <- pedigreeToGraph(pedigree)
#Plot it
plot(pedigreeAsGraph)
#Write it to a file in DOT format
```

---

eightParentPedigreeSingleFunnel

Generate an eight-parent pedigree, using a single funnel

Description

Generate a eight-parent pedigree starting from inbred founders, using a single funnel.

Usage

```
eightParentPedigreeSingleFunnel(
  initialPopulationSize,
  selfingGenerations,
  nSeeds = 1L,
  intercrossingGenerations
)
```

Arguments

initialPopulationSize	The number of initially generated lines, whose genetic material is a mosaic of the eight founding lines. These lines are generated using three generations of structured mating.
selfingGenerations	The number of selfing generations at the end of the pedigree.
nSeeds	The number of progeny taken from each intercrossing line, or from each initially generated line (if no intercrossing is specified). These lines are then selfed according to selfingGenerations.
intercrossingGenerations	The number of generations of random mating performed from the F1 generation. Population size is maintained at that specified by initialPopulationSize.

Value

An object of class `detailedPedigree` representing the experimental design, suitable for simulation using simulateMPCross.

See Also

`eightParentPedigreeSingleFunnel`, `fourParentPedigreeSingleFunnel`, `fourParentPedigreeRandomFunnels`, `twoParentPedigree`

Examples

```
pedigree <- eightParentPedigreeSingleFunnel(initialPopulationSize = 10,
selfingGenerations = 0, nSeeds = 1, intercrossingGenerations = 1)
map <- qtl::sim.map()
cross <- simulateMPCross(map = map, pedigree = pedigree, mapFunction = haldane)
#Get out a list of funnels, which are rows of this matrix. For this pedigree, every funnel is 1:8.
getAllFunnels(cross)
#convert the pedigree to a graph
pedigreeAsGraph <- pedigreeToGraph(pedigree)
#Plot it
plot(pedigreeAsGraph)
#Write it to a file in DOT format
write.graph(graph = pedigreeAsGraph@graph, format = "dot", file = "./pedigree.dot")
```

---

eightParentSubsetMap *Genetic map and genetic data from an 8-parent MAGIC population.*

Description

Genetic map and genetic data from an 8-parent MAGIC population.

Author(s)

Alex Whan, Matthew Morell, Rohan Shah, Colin Cavanagh This dataset contains the genetic map, genetic data, and imputed IBD genotypes for parts of chromosomes 1A, 1B and 1D, from an 8-way MAGIC population of 4229 lines.

---

estimateMap *Estimate map distances*

Description

Estimate map distances based on the estimated recombination fractions

Usage

```
estimateMap(
  mpcrossLG,
  mapFunction = rfToHaldane,
  maxOffset = 1,
  maxMarkers = 2000,
  verbose = FALSE
)
```

Arguments

mpcrossLG	An object of class mpcrossLG, which must also contain data about recombination fractions and linkage groups.
mapFunction	The map function to use to compute recombination fractions to centiMorgan distances.
maxOffset	The maximum separation between pairs of markers used for map construction, in terms of position within the ordering. Recombination fractions between pairs of markers, which are further apart than this, will not be used to estimate the map distances.
maxMarkers	The (approximate) number of markers for which distances are estimated simultaneously.
verbose	Should verbose output be produced?

Details

Once a marker order has been chosen, one possible way of estimating a genetic map is to convert the recombination fractions between adjacent markers into centiMorgan distances. This tends not to work well, because individual recombination fraction estimates can be highly variable, depending on the experimental design used, and the distribution of the marker alleles among the founders. It also wastes much of the information contained in the data; we can estimate recombination fractions between all pairs of markers, rather than just adjacent markers, and this information should be used in the estimation of map distances

This function uses non-linear least squares to estimate map distances as follows. Assume that there are n markers on a chromosome, and for all pairs of markers there is an available estimate of the recombination fraction. For every pair of markers which differ by maxOffset or less, in terms of their position within the ordering, the recombination fraction between these markers is turned into a centiMorgan distance. This centiMorgan distance is expressed as a sum of distances between adjacent markers, which is a simple equation. The set of all the equations generated in this way is represented as a matrix equation, and solved via non-linear least squares. As these non-linear least squares problems can become very large, input maxMarkers allows the non-linear least squares problem to be broken into several smaller problems.

For example, assume that there are five markers, for which an order has been determined. The distance between markers i and j, *as estimated by the recombination fractions*, is $d(i, j)$. The genetic distance between markers i and $i + 1$ *in the final genetic map* is $a(i)$. So in this case, the parameters that are to be estimated are $a(1), a(2), a(3)$ and $a(4)$. If maxOffset is 3, then the set of equations generated is

$$d(1, 3) = a(1) + a(2)$$

$$d(1,4) = a(1) + a(2) + a(3)$$

$$d(2,4) = a(2) + a(3)$$

$$d(3,5) = a(3) + a(4)$$

$$d(2,5) = a(2) + a(3) + a(4)$$

These constraints are represented as a matrix equation and solved for $a(1), a(2), a(3)$ and $a(4)$ using non-linear least squares. However, if `maxOffset` is set to 2, then the set of equations is

$$d(1,3) = a(1) + a(2)$$

$$d(2,4) = a(2) + a(3)$$

$$d(3,5) = a(3) + a(4)$$

Value

A map object, in the format specified by the qtl-package package. This format is a list of chromosomes, with each entry being a named numeric vector of marker positions.

Examples

```
data(simulatedFourParentData)
#Estimate recombination fractions
rf <- estimateRF(simulatedFourParentData)
#Assign all markers to one linkage group / chromosome
grouped <- formGroups(rf, groups = 1)
#Estimate map
estimatedMap <- estimateMap(grouped, maxOffset = 10)
#Create object that includes the map
mapped <- new("mpcrossMapped", grouped, map = estimatedMap)
```

---

estimateMapFromImputation

Re-estimate large gaps in a genetic map from IBD genotype imputations

---

Description

Re-estimate large gaps in a genetic map from IBD genotype imputations

Usage

```
estimateMapFromImputation(
  mpcrossMapped,
  gapSize = 5,
  recombinationFractions = c(0:60/600, 11:49/100)
)
```

Arguments

`mpcrossMapped`	An object of class `mpcrossMapped`
`gapSize`	The size of the gap to reestimate.
`recombinationFractions`	The recombination fractions to use for numerical maximum likelihood estimation

Details

For larger gaps in a genetic map, the pairwise recombination fractions are not (by themselves) useful. An alternative is to estimate the IBD genotypes, and use the imputed IBD genotypes to re-estimate larger gaps using numerical maximum likelihood. Although the IBD genotypes are based on an existing genetic map, they may not be strongly affected by a large gap that has been poorly estimated, as the imputed IBD genotypes represent a consensus across all nearby markers, and also allow for genotyping errors. As a result, the re-estimated map may be different from the original map, and potentially more accurate.

Value

An object of class `mpcrossMapped` with a re-estimated map.

`estimateRF` *Estimate pairwise recombination fractions This function estimates the recombination fractions between all pairs of markers in the input object. The recombination fractions are estimated using numerical maximum likelihood, and a grid search. Because every estimate will be one of the input test values, the estimates can be stored efficiently with a single byte per estimate.*

Description

Estimate pairwise recombination fractions

This function estimates the recombination fractions between all pairs of markers in the input object. The recombination fractions are estimated using numerical maximum likelihood, and a grid search. Because every estimate will be one of the input test values, the estimates can be stored efficiently with a single byte per estimate.

Usage

```
estimateRF(
  object,
  recombValues,
  lineWeights,
  gbLimit = -1,
  keepLod = FALSE,
  keepLkhd = FALSE,
```

```
  verbose = FALSE,
  markerRows = 1:nMarkers(object),
  markerColumns = 1:nMarkers(object)
)
```

Arguments

`object`	An object of class `mpcross`.
`recombValues`	a vector of test values to use for the numeric maximum likelihood step. Must contain 0 and 0.5, and must have less than 255 values in total. The default value is `c(0:20/200,11:50/100)`.
`lineWeights`	Values to use to correct for segregation distortion. This parameter should in general be left unspecified.
`gbLimit`	The maximum amount of working memory this estimation step should be allowed to use at any one time, in gigabytes. Smaller values may increase the computation time. A value of -1 indicates no limit.
`keepLod`	Set to `TRUE` to compute the likelihood ratio score statistics for testing whether the estimate is different from 0.5. Due to memory constraints this should generally be left as `FALSE`.
`keepLkhd`	Set to `TRUE` to compute the maximum value of the likelihood. Due to memory constraints this should generally be left as `FALSE`.
`verbose`	Output diagnostic information, such as the amount of memory required, and the progress of the computation.
`markerRows`	Used to estimate only a subset of the full matrix of pairwise recombination fractions.
`markerColumns`	Used to estimate only a subset of the full matrix of pairwise recombination fractions.

Details

The majority of the options for this function should *not* be specified by the end user. In particular, `keepLkhd`, `keepLod` and `lineWeights` should not be specified without good reason.

Arguments `markerRows` and `markerColumns` can be used to estimate only a subset of the full recombination matrix. Reasons for doing this could include

1. Allowing the full matrix to be estimated in multiple steps, with intermediate computations being saved
2. The matrix of recombination fractions has *mostly* already been estimated. This can occur when adding extra markers.
3. Memory limitations. Performing estimation for markers with many alleles takes a large amount of memory. It is often useful to estimate recombination fractions between all pairs of biallelic markers, and let other pairs be done using a separate call.

If arguments `markerRows` and `markerColumns` are used, only the *upper-triangular part* of the specified subset is computed. See the examples for details.

Value

An object of class mpcrossRF, which contains the original genetic data, and also estimated recombination fraction data.

Examples

```
map <- qtl::sim.map(len = 100, n.mar = 11, include.x=FALSE)
f2Pedigree <- f2Pedigree(1000)
cross <- simulateMPCross(map = map, pedigree = f2Pedigree, mapFunction = haldane, seed = 1)
rf <- estimateRF(cross)
#Print the estimated recombination fraction values
rf@rf@theta[1:11, 1:11]

#Now only estimate recombination fractions between the first 3 markers.
#    The other estimates will just be marked as NA
rf <- estimateRF(cross, markerRows = 1:3, markerColumns = 1:3)
#Print the estimated recombination fraction values
rf@rf@theta[1:11, 1:11]

#A more complicated example, where three values are estimated
rf <- estimateRF(cross, markerRows = 1, markerColumns = 1:3)
#Print the estimated recombination fraction values
rf@rf@theta[1:11, 1:11]

#In this case only ONE value is estimated, because only one element of the requested subset
#    lies in the upper-triangular part - The value on the diagonal.
rf <- estimateRF(cross, markerRows = 3, markerColumns = 1:3)
#Print the estimated recombination fraction values
rf@rf@theta[1:11, 1:11]
```

---

estimateRFSingleDesign

Estimate pairwise recombination fractions

Description

Estimate pairwise recombination fractions, similar to estimateRF, but with different performance requirements in terms of compute time and storage.

Usage

```
estimateRFSingleDesign(
  object,
  recombValues,
  lineWeights,
  keepLod = FALSE,
  keepLkhd = FALSE,
  verbose = FALSE,
```

```
  markerRows = 1:nMarkers(object),
  markerColumns = 1:nMarkers(object)
)
```

Arguments

object	An object of class mpcross.
recombValues	a vector of test values to use for the numeric maximum likelihood step. Must contain 0 and 0.5, and must have less than 255 values in total. The default value is c(0:20/200,11:50/100).
lineWeights	Values to use to correct for segregation distortion. This parameter should in general be left unspecified.
keepLod	Set to TRUE to compute the likelihood ratio score statistics for testing whether the estimate is different from 0.5. Due to memory constraints this should generally be left as FALSE.
keepLkhd	Set to TRUE to compute the maximum value of the likelihood. Due to memory constraints this should generally be left as FALSE.
verbose	Output diagnostic information, such as the amount of memory required, and the progress of the computation.
markerRows	Used to estimate only a subset of the full matrix of pairwise recombination fractions.
markerColumns	Used to estimate only a subset of the full matrix of pairwise recombination fractions.

Details

Estimate pairwise recombination fractions, similar to estimateRF, but with different performance requirements in terms of compute time and storage. Specifically, this version is expected to perform better when there is only a single population.

Value

An object of class mpcrossRF, which contains the original genetic data, and also estimated recombination fraction data.

---

existingLocalisationStatistics

Localisation statistics for example of callFromMap

Description

This dataset contains the localisation statistics for the example for running callFromMap. This makes the example fast enough to pass the CRAN check.

expand *Expand markers contained within object*

Description

Expand set of markers within object, adding extra markers with missing observations as necessary.

Usage

```
expand(mpcross, newMarkers)
```

Arguments

mpcross	The input mpcross object
newMarkers	The names of the new markers to add

Details

This function expands the set of markres within an mpcross object. The new set of marker names must contain all the existing marker names, with any desired extra marker names. Any added markers will have all observations marked as missing. Any existing non-genetic information (genetic map, assignment of linkage groups, IBD genotypes, IBD probabilitieS) will be removed.

Value

An object of class mpcross with a larger set of markers.

exportMapToPretzl *Export genetic map to Pretzl*

Description

Export genetic map to Pretzl

Usage

```
exportMapToPretzl(inputObject, name, separateChromosomes = FALSE)
```

Arguments

inputObject	The object of class mpcrossMapped containing the genetic map
name	If a single JSON object is being exported, the name of the exported map.
separateChromosomes	If TRUE, separate exports will be generated for each chromosome. The name associated with each chromosome map will contain the chromosome name as a suffix.

Details

Convert the genetic map from an object of class `mpcrossMapped` to the JSON format used by Pretzl. Pretzl is a web app for visualising and comparing genetic maps.

Value

A list containing JSON, suitable for import into Pretzl.

`extraImputationPoints` *Get out non-marker positions used for IBD genotype imputation*

Description

Get out non-marker positions used for IBD genotype imputation

Usage

```
extraImputationPoints(mpcrossMapped)
```

Arguments

`mpcrossMapped`	The object from which to get the non-marker positions

Details

Extract non-marker positions used for IBD genotype imputation

Value

A vector of genetic position names.

`f2Pedigree` *Generate an F2 pedigree which starts from inbred founders*

Description

Generate an F2 pedigree which starts from inbred founders

Usage

```
f2Pedigree(populationSize)
```

Arguments

`populationSize`	The size of the generated population.

Value

An object of class `detailedPedigree` representing the experimental design, suitable for simulation using simulateMPCross.

Examples

```
pedigree <- f2Pedigree(1000)
#This pedigree is automatically marked as involving finite generations of selfing.
selfing(pedigree)
```

`finalNames` *Names of genetic lines Return the names of the genetic lines If the* `mpcross` *object contains a single experiment a vector of names of genetic lines is returned. The names of the founding lines for the population are excluded. If an* `mpcross` *object contains multiple experiments a list of vectors of names is returned.*

Description

Names of genetic lines

Return the names of the genetic lines

If the `mpcross` object contains a single experiment a vector of names of genetic lines is returned. The names of the founding lines for the population are excluded. If an `mpcross` object contains multiple experiments a list of vectors of names is returned.

Usage

```
finalNames(object)

## S4 method for signature 'mpcross'
finalNames(object)

## S4 method for signature 'geneticData'
finalNames(object)
```

Arguments

`object`	The `mpcross` object from which to extract the names of the genetic lines

Value

The names of the genetic lines in the final population.

`finals` *Genetic data for final lines Return the genetic data matrix for the final lines If the* `mpcross` *object contains a single experiment a matrix is returned, with rows corresponding to genotyped lines and columns corresponding to markers. The founding lines of the population are excluded from this matrix. If an* `mpcross` *object contains multiple experiments a list of such matrices is returned, one for each experiment.*

Description

Genetic data for final lines

Return the genetic data matrix for the final lines

If the `mpcross` object contains a single experiment a matrix is returned, with rows corresponding to genotyped lines and columns corresponding to markers. The founding lines of the population are excluded from this matrix. If an `mpcross` object contains multiple experiments a list of such matrices is returned, one for each experiment.

Usage

```
finals(object)

## S4 method for signature 'mpcross'
finals(object)

## S4 method for signature 'geneticData'
finals(object)
```

Arguments

`object`	The `mpcross` object from which to extract the genetic data matrix

Value

An integer matrix with rows corresponding to genotyped lines and columns corresponding to markers.

`fixedNumberOfFounderAlleles`
Convert fully informative experiment to one with a fixed number of alleles per marker

Description

Convert a fully informative experiment to one with a fixed number of alleles per marker

Usage

```
fixedNumberOfFounderAlleles(alleles)
```

Arguments

alleles	Number of alleles for each marker

Details

By default, simulated data is fully informative, so every founder carries its own allele, and all heterozygotes are distinguishable.

This function takes in a fully informative experiment, and changes every marker so that it has a fixed number of founder alleles. Heterozygotes are also changed, so every combination of different alleles is still distinguishable.

Value

An object of internal class `fixedNumberOfFounderAlleles` suitable for application to an object of class `mpcross` using the addition operation.

Examples

```
data(simulatedFourParentData)
founders(simulatedFourParentData)[, 1:10]
altered <- simulatedFourParentData + fixedNumberOfFounderAlleles(3)
founders(altered)[, 1:10]
```

---

flatImputationMapNames

Get names of positions for IBD genotype imputation

Description

Get the names of all positions at which IBD genotype imputation has already been performed

Usage

```
flatImputationMapNames(object, ...)

## S4 method for signature 'imputed'
flatImputationMapNames(object, ...)

## S4 method for signature 'geneticData'
flatImputationMapNames(object, ...)

## S4 method for signature 'mpcrossMapped'
flatImputationMapNames(object, ...)
```

Arguments

object	The object from which to get the names of positions
...	Extra parameters, currently only "experiment" is supported.

Details

Get the names of all positions at which IBD genotype imputation has already been performed

Value

The names of all positions at which IBD genotype imputation has already been performed.

---

formGroups *Form linkage groups*

Description

Group markers into linkage groups using hierarchical clustering.

Usage

```
formGroups(
  mpcrossRF,
  groups,
  clusterBy = "theta",
  method = "average",
  preCluster = FALSE
)
```

Arguments

mpcrossRF	An object of class mpcrossRF.
groups	The number of groups to form
clusterBy	The matrix to use for clustering. The three choices are theta (recombination fractions), lod (log-odds ratio) or combined (a combination of both).
method	The method to use for hierarchical cluster. Choices are average, complete and single.
preCluster	Before clustering is performed, should we form groups of markers which are completely linked?

Details

This function groups markers into the specified number of linkage groups, using hierarchical clustering. This can be done using three different dissimilarity matrices, specified by the `clusterBy` argument. If `"theta"` is specified, then the matrix of recombination fractions is used. If `"lod"` is specified, then a matrix of likelihood ratio test statistics is used. The hypothesis being tested is whether the recombination fraction is 0.5 (no linkage). If `"combined"` is specified, then a combination of both previous approaches is used. We recommend the default value of `"theta"`.

The linkage method for hierachical clustering is specified by the `method` argument; acceptable values are `"average"`, `"complete"` and `"single"`.

Argument `preCluster` determines whether the code combines markers that are completely linked, before performing hierarchical clustering. This can lead to speed-ups in clustering truly huge datasets.

Value

An object of class `mpcrossLG`, containing all the information in the input object and also information about linkage groups.

`founderNames` *Names of founding genetic lines Return the names of the founding genetic lines If the* `mpcross` *object contains a single experiment a vector of names of genetic lines is returned. If an* `mpcross` *object contains multiple experiments a list of vectors of names is returned.*

Description

Names of founding genetic lines

Return the names of the founding genetic lines

If the `mpcross` object contains a single experiment a vector of names of genetic lines is returned. If an `mpcross` object contains multiple experiments a list of vectors of names is returned.

Usage

```
founderNames(object)

## S4 method for signature 'mpcross'
founderNames(object)

## S4 method for signature 'geneticData'
founderNames(object)
```

Arguments

`object`	The `mpcross` object from which to extract the names of the founding genetic lines

Value

A vector of names of genetic lines, or a list of such vectors, in the case of multiple experiments.

`founders` *Genetic data for founding lines Return the genetic data matrix for the founding lines If the* `mpcross` *object contains a single experiment a matrix is returned, with rows corresponding to founding lines and columns corresponding to markers. If an* `mpcross` *object contains multiple experiments a list of such matrices is returned, one for each experiment.*

Description

Genetic data for founding lines

Return the genetic data matrix for the founding lines

If the `mpcross` object contains a single experiment a matrix is returned, with rows corresponding to founding lines and columns corresponding to markers. If an `mpcross` object contains multiple experiments a list of such matrices is returned, one for each experiment.

Usage

```
founders(object)

## S4 method for signature 'mpcross'
founders(object)

## S4 method for signature 'geneticData'
founders(object)
```

Arguments

`object`	The `mpcross` object from which to extract the genetic data matrix of the founding lines

Value

An integer matrix, with rows corresponding to founding lines and columns corresponding to markers, or a list of such matrices in the case of multiple experiments.

fourParentPedigreeRandomFunnels
Generate a four-parent pedigree

Description

Generate a four-parent pedigree starting from inbred founders, using a random funnel

Usage

```
fourParentPedigreeRandomFunnels(
  initialPopulationSize,
  selfingGenerations,
  nSeeds = 1L,
  intercrossingGenerations
)
```

Arguments

initialPopulationSize	The number of F1 lines generated
selfingGenerations	The number of selfing generations at the end of the pedigree
nSeeds	The number of progeny taken from each intercrossing line, or from each F1 if no intercrossing is specified. These lines are then selfed according to selfingGenerations
intercrossingGenerations	The number of generations of random mating performed from the F1 generation. Population size is maintained at that specified by initialPopulationSize

Value

An object of class `detailedPedigree` representing the experimental design, suitable for simulation using simulateMPCross.

See Also

`fourParentPedigreeSingleFunnel`, `twoParentPedigree`

`fourParentPedigreeSingleFunnel` *Generate a four-parent pedigree*

Description

Generate a four-parent pedigree starting from inbred founders, using a single funnel

Usage

```
fourParentPedigreeSingleFunnel(
  initialPopulationSize,
  selfingGenerations,
  nSeeds = 1L,
  intercrossingGenerations
)
```

Arguments

`initialPopulationSize`	The number of F1 lines generated
`selfingGenerations`	The number of selfing generations at the end of the pedigree
`nSeeds`	The number of progeny taken from each intercrossing line, or from each F1 if no intercrossing is specified. These lines are then selfed according to selfingGenerations
`intercrossingGenerations`	The number of generations of intercrossing, after each F2 line is generated.

Details

Note that unlike `fourParentPedigreeRandomFunnels`, there is no intercrossing allowed in the single funnel case because the relevant haplotype probabilities assume randomly chosen funnels

Value

An object of class `detailedPedigree` representing the experimental design, suitable for simulation using simulateMPCross.

See Also

`fourParentPedigreeRandomFunnels`, `twoParentPedigree`

fromMpMap *Convert from mpMap format to mpMap2 format*

Description

Convert an object from mpMap format into mpMap2 format

Usage

```
fromMpMap(mpcross, selfing = "infinite", fixCodingErrors = FALSE)
```

Arguments

mpcross	Object to convert
selfing	Number of generations of selfing to put in the pedigree, for the converted object. Must be "finite" or "infinite".
fixCodingErrors	Should we attempt to fix coding errors, by replacing invalid values by NA? Defaults to FALSE.

Details

Convert an object from mpMap format (the predecessor to mpMap2) into mpMap2 format. It is unlikely that this function will ever need to be used.

Value

An object of class mpcross or mpcrossMapped, depending on the data contained in the input object.

generateGridPositions *Specify an equally spaced grid of genetic positions*

Description

Specify an equally spaced grid of genetic positions

Usage

```
generateGridPositions(spacing)
```

Arguments

spacing	The spacing of the genetic positions, in cM.

Details

Some functions, such as `imputeFounders` and `computeGenotypeProbabilities`, take in a set of genetic positions as one of the inputs. This function is an easy way to specify an equally spaced grid of positions.

Note that the return value is itself a function, which is applied internally by `imputeFounders` or `computeGenotypeProbabilities` to an object of class `mpcrossMapped`.

Value

A function which can be applied to an object of class `mpcrossMapped` by `imputeFounders` or `computeGenotypeProbabilities`.

Examples

```
data(simulatedFourParentData)
#Create object that includes the correct map
mapped <- new("mpcrossMapped", simulatedFourParentData, map = simulatedFourParentMap)
#Estimate IBD genotypes at all the markers, and marker midpoints
imputed <- imputeFounders(mapped, errorProb = 0.02,
extraPositions = generateGridPositions(1))
```

`generateIntervalMidPoints` *Specify interval midpoints*

Description

Specify interval midpoints

Usage

```
generateIntervalMidPoints(object)
```

Arguments

`object`	The object of class `mpcrossMapped` from which to take the interval midpoints.

Details

Some functions, such as `imputeFounders` and `computeGenotypeProbabilities`, take in a set of genetic positions as one of the inputs. This function is an easy way to specify the midpoint of every marker interval.

Note that you don't have to explicitly evaluate this function, it can be passed in directly (see examples).

Value

A function which can be applied to an object of class `mpcrossMapped` by `imputeFounders` or `computeGenotypeProbabilities`.

Examples

```
data(simulatedFourParentData)
#Create object that includes the correct map
mapped <- new("mpcrossMapped", simulatedFourParentData, map = simulatedFourParentMap)
#Estimate IBD genotypes at all the markers, and marker midpoints
imputed <- imputeFounders(mapped, errorProb = 0.02,
extraPositions = generateIntervalMidPoints(mapped))
#Alternatively we can explicitly evaluate the function. This is identical to above.
imputed <- imputeFounders(mapped, errorProb = 0.02,
extraPositions = generateIntervalMidPoints)
```

`geneticData-class` *Object containing the genetic data for a population*

Description

Object containing the genetic data for a population

Details

This object contians the genetic data for a population. Required data includes the genetic data for the founding lines of the poulation, the final lines of the population, information about the enoding of heterozygotes, and the pedigree used to generate the final genetic lines from the founding genetic line.

Optional data includes IBD genotype imputations, a data.frame of phenotypes, and IBD genotype probabilities.

This class has extensive validity checking, to ensure that all the different inputs are compatible and meet the requirements. If an error is found, an informative error message should be produced.

Slots

`founders` The genetic data for the founding lines of the population. Must be an integer matrix, where rows correspond to genetic lines and columns correspond to genetic markers.

`finals` The genetic data for the final lines of the population. Must be an integer matrix, where rows correspond to genetic lines and columns correspond to genetic markers.

`hetData` Information about the encoding of marker heterozygotes.

`pedigree` Object of class `pedigree` with information about how the final genetic lines are generated from the founding lines.

`imputed` Optional data about imputed IBD genotypes. Can be generated using `imputeFounders`, assuming there is a genetic map available.

probabilities Optional data about IBD genotype probabilities. Can be generated using computeGenotypeProbabilities assuming there is a genetic map available.

pheno Optional

---

getAllFunnels *Get funnels*

Description

Get the order of the founding lines, as they contribute to each line in the final population

Usage

```
getAllFunnels(cross, standardised = FALSE)
```

Arguments

cross	The object of class mpcross containing the pedigree of interest
standardised	Should the output funnels be standardised?

Details

In multi-parent experimental designs, the founding lines of the population are combined together through initial mixing generations. For experiments without further intercrossing generations, the order in which these mixing crosses occur influences the genotypes of the final lines. It can be important to examine or visualise these orders, which are known as funnels.

This function returns a matrix, where each row corresponds to a genetic line in the final population, and each column corresponds to a position in the mixing step. So if a row of the returned matrix contains the values 4, 1, 2, 3, then the pedigee that generated the first individual in the experiment started by crossing founders 4 and 1 to give individual 41, and 2 and 3 to give individual 23. Then individuals 41 and 23 are crossed to generate individual 4123, which after inbreeding results in the first final genetic line.

If sex is considered to be unimportant, then many orderings are equivalent. For example, the ordering 4, 1, 2, 3 of the initial founders is equivalent to 1, 4, 2, 3. In this case each funnel can be put into a standardised ordering, by setting standardised to FALSE.

Note that if there are generations of random interbreeding in the population (often referred to as maintenance generations), then there is no "funnel" associated with a genetic line, and values of NA are returned. In that case, see getAllFunnelsIncAIC.

Note that funnels for all pedigrees simulated by mpMap2 are already standardised. This will not generally be the case for realy experiments.

Value

An integer matrix with rows representing genetic lines, and columns representing positions within the funnel.

Examples

```
data(simulatedFourParentData)
#Funnels used to generate the first ten lines
#Because this is simulated data, they are already standardised,
#' with the first founder in the first position in the mixing step.
getAllFunnels(simulatedFourParentData)[1:10, ]
```

`getAllFunnelsIncAIC` *Get all funnels, including AIC lines*

Description

Get every order of the founding lines, which makes a contribution to the final population

Usage

```
getAllFunnelsIncAIC(cross, standardised = FALSE)
```

Arguments

`cross`	The object of class `mpcross` containing the pedigree of interest
`standardised`	Should the output funnels be standardised?

Details

This function is similar to `getAllFunnels`, but more useful for populations with maintenance (or AIC) generations. It returns a list of all the mixing orders in the initial generations, which make a genetic contribution to the final population. Unlike for `getAllFunnels`, rows of the returned matrix DO NOT refer to specific genetic lines.

Value

Matrix of mixing orders that contribute to the final popluation. Rows DO NOT refer to specific genetic lines.

Examples

```
set.seed(1)
pedigree <- fourParentPedigreeRandomFunnels(initialPopulationSize = 1000,
     selfingGenerations = 6, intercrossingGenerations = 1)
#Assume infinite generations of selfing in subsequent analysis
selfing(pedigree) <- "infinite"
#Generate random map
map <- qtl::sim.map(len = 100, n.mar = 101, anchor.tel = TRUE, include.x = FALSE)
#Simulate data
cross <- simulateMPCross(map = map, pedigree = pedigree, mapFunction = haldane, seed = 1L)
#Because we have maintenance in this experiment, we can't get out the funnels per genetic line
funnels <- getAllFunnels(cross)
```

```
dim(funnels)
funnels[1:10,]
#But we can get out a list of all the funnels that go into the experiment.
funnels <- getAllFunnelsIncAIC(cross)
dim(funnels)
funnels[1:10,]
```

`getChromosomes` *Get chromosome assignment per marker*

Description

Get chromosome assignment per marker from an `mpcross` object.

Usage

```
getChromosomes(mpcrossMapped, markers)
```

Arguments

`mpcrossMapped`	The object containing the map of interest
`markers`	The markers for which we want the chromosomes

Details

Extract a character vector, with names corresponding to markers, and values corresponding to the chromosome on which the named marker is located.

Value

A character vector, with names corresponding to markers, and values corresponding to the chromosome on which the named marker is located.

Examples

```
map <- qtl::sim.map()
pedigree <- f2Pedigree(1000)
cross <- simulateMPCross(map = map, pedigree = pedigree, mapFunction = haldane, seed = 1)
mappedCross <- mpcrossMapped(cross = cross, map = map)
chromosomeAssignment <- getChromosomes(mappedCross, markers(mappedCross))
chromosomeAssignment
```

getIntercrossingAndSelfingGenerations *Identify number of generations of intercrossing and selfing, per genetic line*

Description

Identify number of generations of intercrossing and selfing, per genetic line

Usage

```
getIntercrossingAndSelfingGenerations(cross)
```

Arguments

cross	The mpcross object containing the pedigree to be analysed.

Details

Many structured populations consist of a number of generations of mixing, followed by a number of generations of intercrossing, followed by inbreeding. This function identifies the number of generations of selfing and intercrossing, for each genetic line, in the case of 4-way, 8-way or 16-way multi-parent design.

Value

An integer matrix with two columns, giving the number of generations of selfing and intercrossing, for each genetic line. Or in the case of multiple experiments contained within a single object, a list of such matrices.

getPositions *Get positions of genetic markers*

Description

Get positions of genetic markers, on their respective chromosomes

Usage

```
getPositions(mpcrossMapped, markers)
```

Arguments

mpcrossMapped	The mpcross object containing the map of interest
markers	The markers for which to get the positions

Details

Get positions of genetic markers in cM, on their respective chromosomes

Value

A named vector of numbers, with names corresponding to the selected genetic markers, and values corresponding to genetic positions.

Examples

```
map <- qtl::sim.map()
pedigree <- f2Pedigree(1000)
cross <- simulateMPCross(map = map, pedigree = pedigree, mapFunction = haldane, seed = 1)
mappedCross <- mpcrossMapped(cross = cross, map = map)
getPositions(mappedCross, c("D13M3", "DXM1", "DXM3"))
```

---

hetData *Get the encoding of marker heterozygotes*

Description

Get the encoding of marker heterozygotes

Usage

```
hetData(object, marker)

## S4 method for signature 'mpcross'
hetData(object, marker)

## S4 method for signature 'geneticData'
hetData(object, marker)
```

Arguments

object	The object from which to extract the encoding data
marker	The marker of interest. If this is missing, heterozygote encoding data is returned for all markers.

Details

Get the encoding of markers heterozygotes, either for all markers, or a specific marker.

Value

Heterozygote encoding data, for either a specific marker or all markers.

hetsForSNPMarkers *Create heterozygote encodings for SNP markers*

Description

Create encoding which assumes that the single non-homozygote value for a SNP marker is the heterozygote

Usage

```
hetsForSNPMarkers(founders, finals, pedigree)
```

Arguments

founders	Genetic data for the founding lines of the population
finals	Genetic data for the final genotyped lines of the population
pedigree	Pedigree for the population. Unused by this particular function.

Details

This function takes in genotype data for the founding lines and the final poulation. It returns an encoding for hetorozygotes for all markers, where multiallelic markers are assumed to have no heterozygotes. For biallelic markers with three observed alleles in the final population, the extra allele is assumed to be the heterozygote.

Value

An object of class hetData, which contains encodings for the marker heterozygotes and the (unique) marker heterozygote

imputationData *Get out the IBD genotype imputation data*

Description

Get out the IBD genotype imputation data

Usage

```
imputationData(object, ...)

## S4 method for signature 'imputed'
imputationData(object, ...)

## S4 method for signature 'geneticData'
imputationData(object, ...)

## S4 method for signature 'mpcrossMapped'
imputationData(object, ...)
```

Arguments

`object`	The object from which to extract the IBD genotype imputation data
`...`	Extra parameters. Currently only `"experiment"` is supported, letting the user extract the imputation data for a specific experiment.

Details

Extract the IBD genotype imputation data. The data takes the form of a matrix of values, with rows corresponding to genetic lines and columns corresponding to genetic positions. The genetic positions may include non-marker positions, so use `imputationMap` to find out the chromosome and position for every marker.

Each value in the matrix represents the predicted genotype for that genetic line, at that position. In the case of completely inbred experiments, each value in the matrix represents the founders from which that allele is believed to be derived. In the case of experiments with residual heterozygosity, the possible genotypes include heterozygotes, and the interpretation of the values in the matrix is more complicated. Function `imputationKey` gives information about how the values in the matrix correspond to actual genotypes.

Value

The IBD genotype imputation data.

`imputationKey` *Get out key for IBD genotype imputations*

Description

Get out key for IBD genotype imputations

Usage

```
imputationKey(object, ...)

## S4 method for signature 'imputed'
imputationKey(object, ...)

## S4 method for signature 'geneticData'
imputationKey(object, ...)

## S4 method for signature 'mpcrossMapped'
imputationKey(object, ...)
```

Arguments

object	The object from which to get the imputation key.
...	Extra parameters. Currently only "experiment" is supported, letting the user extract the imputation map for a specific experiment.

Details

When IBD genotype imputation is performed using a population with finite generations of selfing, some of the imputed genotypes will be heterozygotes. However, the imputation code only returns a single value per line per genetic position. This key translates that value to a pair of founder alleles.

The key is a matrix with three columns. The first two columns represent founder alleles, and the third column gives the encoding for that particular pair of founder alleles.

Value

Key giving the encoding of heterozygotes, in the imputed IBD genotype data.

Examples

```
pedigree <- eightParentPedigreeRandomFunnels(initialPopulationSize = 100,
selfingGenerations = 2, nSeeds = 1, intercrossingGenerations = 0)
selfing(pedigree) <- "finite"
#Generate map
map <- qtl::sim.map()
#Simulate data
cross <- simulateMPCross(map = map, pedigree = pedigree, mapFunction = haldane)
crossSNP <- cross + multiparentSNP(keepHets = TRUE)
crossMapped <- mpcrossMapped(crossSNP, map = map)
imputed <- imputeFounders(crossMapped, errorProb = 0.01)
#An imputed IBD genotype of 1 indicates a homozygote for founder 1
#An imputed IBD genotype of 9 indicates a heterozygote for founders 1 and 2
#etc
head(imputationKey(imputed))
```

imputationMap *Get map used for IBD genotype imputation*

Description

Get map used for IBD genotype imputation

Usage

```
imputationMap(object, ...)

## S4 method for signature 'imputed'
imputationMap(object, ...)

## S4 method for signature 'geneticData'
imputationMap(object, ...)

## S4 method for signature 'mpcrossMapped'
imputationMap(object, ...)
```

Arguments

object	The object from which to extract the IBD genotype imputation positions.
...	Extra parameters. Currently only "experiment" is supported, letting the user extract the imputation map for a specific experiment.

Details

Get the map of positions used for IBD genotype imputation. This is necessary because the points at which IBD genotype imputation has been performed may include non-marker points. See imputeFounders for further details.

Value

The map of positions used for IBD genotype imputation.

impute *Impute missing recombination fraction estimates*

Description

Impute missing recombination fraction estimates

Usage

```
impute(
  mpcrossLG,
  verbose = FALSE,
  allErrors = FALSE,
  extractErrorsFunction = function(e) e
)
```

Arguments

mpcrossLG	An object of class mpcrossLG, which contains estimated pairwise recombination fractions
verbose	Should more verbose output be generated?
allErrors	If there is an error, should we immediately return, or should we continue, and report all errors?
extractErrorsFunction	Error handling function. If there are errors and allErrors is TRUE, this function will be called with a matrix indicating which estimates could not be imputed.

Details

Recombination fractions between every pair of markers are estimated using numerical maximum likelihood. Unfortunately the likelihood is flat in some cases, so an estimate cannot be made. This later causes problems when trying to use estimated recombination fractions to order the markers, because a complete matrix of estimates is required. The solution is to impute the missing estimates using related estimates. For example, the recombination fraction between markers A and C may not be directly estimatable. However, there may be a marker B known to be tightly linked to A, which has a known recombination fraction with C. The estimated recombination fraction between B and C can be taken to be an estimate of the recombination fraction between A and C.

This function imputes values in the estimated recombination fraction matrix, to return a complete matrix. If there is a value that cannot be imputed, an error is triggered. Input allErrors controls whether the function will stop after encountering a single error, or continue and report all errors. If all errors are being reported, the optional function extractErrorsFunction is called with information about which missing estimates could not be imputed.

Value

An object of class mpcrossLG, containing all the information in the input object, but also an imputed copy of the estimated recombination fraction data.

---

imputeFounders *Impute underlying genotypes*

Description

Impute the most likely sequence of underlying genotypes, using the Viterbi algorithm

Usage

```
imputeFounders(
  mpcrossMapped,
  homozygoteMissingProb = 1,
  heterozygoteMissingProb = 1,
  errorProb = 0,
  extraPositions = list(),
  showProgress = FALSE
)
```

Arguments

mpcrossMapped	An object containing genetic data and a genetic map
homozygoteMissingProb	The probability with which homozygous genotypes are observed as missing.
heterozygoteMissingProb	The probability with which heterozygous genotypes are observed as missing.
errorProb	The probability of a genotyping error.
extraPositions	Extra genetic positions at which to perform imputation.
showProgress	If this paramater is TRUE, a progress bar is produced.

Value

An object of class mpcrossMapped, containing all the information in the input object, and also including imputed IBD genotypes. This function uses the Viterbi algorithm to calculate the most likely sequence of underlying genotypes, given observed genetic data. The parameters for the algorithm are a homozygous mising rate, a heterozygous missing rate, and an error probability.

The two missing rates are intended to allow long strings of missing values to be imputed as heterozygotes, in the case that heterozygous genotypes are observed as missing much more often than homozygotes. Only the ratio of these two parameters is relevant, which is why the default values of 1 are acceptable. These default values really mean that the missing rates are equal.

The parameter extraPositions specifies the genetic positions at which imputation should be performed. This can be either a list, or a function such as generateGridPositions generateIntervalMidPoints. If a function is input, this function is applied to the input genetic map, to determine the extra genetic locations. If a list is input, the names of the list entries should be chromosome names, and the entry for each chromosome should be a named vector. We give an example of the list format in the examples section at the bottom of this page.

One subtlety when using extra genetic positions is that specifying such positions can change the results of the imputation process. This is undesirable, but does not represent a bug in the implementation. The Hidden Markov Model (HMM) used to model the genotypes is not exact, although it is a highly accurate approximation. As it is an approximation, it fails to satisfy the condition

$$P^{s+t} = P^t P^s$$

This property (a stochastic semigroup property) fails to hold because the HMM is only an approximation. As a result, adding extra genetic positions can change the results of the imputation. We

emphasise that this is possible only when there are number of underlying sequences which are almost equally likely, and even then this problem occurs rarely. However, this problem becomes obvious when large simulation studies are performed.

`infiniteSelfing` *Create allele encoding corresponding to infinite generations of selfing*

Description

Create allele encoding corresponding to infinite generations of selfing

Usage

```
infiniteSelfing(founders, finals, pedigree)
```

Arguments

`founders`	The genetic data for the founding lines, which are assumed to be inbred
`finals`	The genetic data for the lines genotyped at the end of the experiment.
`pedigree`	The pedigree for the experiment

Details

In many experiments (particularly those that are significantly inbred), only marker homozygotes are observed, which means that the relationship between marker genotypes and marker alleles is particularly simple. In such cases, generally a marker genotype of some value (say 0) indicates that the individual is homozygous for marker allele 0.

This function takes in genetic data for the founding lines, genetic data for the final population, and the pedigree. It returns an encoding for marker genotypes where every genotype is homozygous for the marker allele with the same value.

Value

An object of class `hetData`, which encodes only the marker homozygotes.

Examples

```
map <- qtl::sim.map()
pedigree <- f2Pedigree(1000)
cross <- simulateMPCross(map = map, pedigree = pedigree, mapFunction = haldane, seed = 1)
#Initially the object contains markers that are fully informative.
#The final genetic data contains values 1, 2 and 3, while the genetic data for the founding
#    lines contains only values 1 and 2.
#A value of 1 or 2 in the final genetic data indicates a homozygote for the
# corresponding marker allele.
#A value of 3 in the final genetic data indicates a heterozygote for the marker allele.
#Information about this encoding is stored in the hetData slot.
hetData(cross, "D1M1")
```

```
cross <- cross + biparentalDominant()
#Now we have converted all markers to dominant.
#The final genetic data contains values 1 and 2, and the genetic data for the founding
#    lines contains only values 1 and 2.
#A value of 2 indicates a homozygote for the corresponding marker allele, OR a
# marker heterozygote.
hetData(cross, "D1M1")
#But under infinite generations of selfing, the encoding is simpler.
simpleEncoding <- infiniteSelfing(founders = founders(cross), finals = finals(cross),
pedigree = pedigree)
simpleEncoding[["D1M1"]]
```

`initialize,canSkipValidity-method`

Initialize method which can skip the validity check

Description

This is an initialization method with an optional `skipValidity` argument. If this argument is set to `TRUE`, the validity check is skipped. This is used by some internal functions within the package, as the validity check can be slow, and internal code is (presumably) guaranteed to produce valid objects.

Usage

```
## S4 method for signature 'canSkipValidity'
initialize(.Object, ...)

## S4 method for signature 'geneticDataList'
initialize(.Object, ...)
```

Arguments

`.Object`	the object to initialize
`...`	Other arguments. Only `skipValidity` is used.

Details

Initialize method which can skip the validity check

jitterMap *Add noise to marker positions*

Description

Add noise to marker positions, so that no markers are co-located

Usage

```
jitterMap(map)
```

Arguments

map	The map to add noise to.

Details

Add noise to marker positions, so that no markers are located at the same position on a single chromosome. This was necessary before there was an error model implemented in the IBD genotype imputation and IBD genotype probabliity code. There is little reason to use this function now.

Value

A copy of the input map, with noise added to genetic positions.

lineNames *Get or set the genetic line names*

Description

Get or set the genetic line names associated with a pedigree or `mpcross` object.

Usage

```
lineNames(object)
```

Arguments

object	The object from which to extract the line names

Details

These functions get or set the names of the genetic lines associated with a pedigree or `mpcross` object.

Value

Vector of genetic line names

lineNames,mpcross-method *Get the genetic line names*

Description

Get the genetic line names of a population

Usage

```
## S4 method for signature 'mpcross'
lineNames(object)

## S4 method for signature 'geneticData'
lineNames(object)
```

Arguments

object	The object from which to extract the line names

Details

These functions get the names of the genetic lines associated with an mpcross object.

Value

Vector of genetic line names

lineNames<- *Get or set the genetic line names of a pedigree*

Description

Get or set the genetic line names of a pedigree

Usage

```
lineNames(object) <- value

## S4 method for signature 'pedigree'
lineNames(object)

## S4 replacement method for signature 'detailedPedigree'
lineNames(object) <- value

## S4 replacement method for signature 'pedigree'
lineNames(object) <- value
```

Arguments

object	The object for which to get or set the line names
value	The vector of new genetic line names

Details

These functions get or set the names of the genetic lines associated with a pedigree.

Value

None

---

linesByNames *Extract pedigree by names*

Description

Extract part of pedigree in human-readable format

Usage

```
linesByNames(pedigree, names)
```

Arguments

pedigree	An object of class pedigree
names	The names of the lines for which to extract the pedigree

Details

Pedigrees in mpMap2 are stored using indices for maternal and paternal lines, which is not a human-readable format. This function takes in a pedigree, and returns a human-readable subset.

Value

A matrix giving the genetic lines and their parents, by line name.

`listCodingErrors` *Generate a list of encoding errors*

Description

Generate a list of encoding errors from genetic data

Usage

```
listCodingErrors(founders, finals, hetData)
```

Arguments

`founders`	Genetic data for the founding lines of the population
`finals`	Genetic data for the final lines of the population
`hetData`	Data about the encoding of marker heterozygotes

Details

Given genetic data matrices for the founding lines and the final lines of a population, and information about the encoding of marker heterozygotes, generate a list of errors. These errors include observed values which don't correspond to a known combination of marker alleles, missing values in the genetic data for the founding lines, etc.

The results of this function allow human-readable lists of errors to be generated, or errors to be automatically fixed (if the errors are sufficiently simple).

Value

List with the following entries:

finals Markers with an invalid observed value.

null Markers with a missing value for a founding line, for which the are observations for at least one genetic line.

missingHetData Markers for which a homozygote did not have an encoding.

invalidHetData Markers for which the heterozygote encoding data was invalid.

listCodingErrorsInfiniteSelfing *Generate a list of encoding errors assuming infinite selfing*

Description

Generate a list of encoding errors assuming infinite selfing

Usage

```
listCodingErrorsInfiniteSelfing(founders, finals)
```

Arguments

founders	Genetic data for the founding lines of the population
finals	Genetic data for the final lines of the population

Details

Generate a list of encoding errors assuming infinite selfing. Given the infinite selfing assumption, no information about heterozygote encoding is required.

Value

List with the following entries:

finals Markers with an invalid observed value.

null Markers with a missing value for a founding line, for which the are observations for at least one genetic line.

missingHetData Markers for which a homozygote did not have an encoding.

invalidHetData Markers for which the heterozygote encoding data was invalid.

mapFunctions *Map functions*

Description

Functions used to convert between recombination fractions and centiMorgan distances.

Usage

```
haldaneToRf(x)

haldane(x)

rfToHaldane(r)

rfToKosambi(r)

kosambiToRf(x)

kosambi(x)
```

Arguments

x	centiMorgan distance
r	recombination fraction

Value

Recombination fraction.

Genetic distance in cM.

Genetic distance in cM.

Recombination fraction.

Recombination fraction.

Functions

- `haldaneToRf`: Convert from Haldane distance to recombination fraction
- `haldane`: Convert from Haldane distance to recombination fraction
- `rfToHaldane`: Convert from recombination fraction to Haldane distance
- `rfToKosambi`: Convert from recombination fraction to Kosambi distance
- `kosambiToRf`: Convert from Kosambi distance to recombination fraction
- `kosambi`: Convert from recombination fraction to Kosambi distance

---

`markers` *Genotyped markers Return the names of the genotyped markers. If an* `mpcross` *object contains multiple experiments, all experiments are required to have the same markers. So a single vector of marker names is returned, in all cases.*

Description

Genotyped markers

Return the names of the genotyped markers.

If an mpcross object contains multiple experiments, all experiments are required to have the same markers. So a single vector of marker names is returned, in all cases.

Usage

```
markers(object)

## S4 method for signature 'mpcross'
markers(object)

## S4 method for signature 'geneticData'
markers(object)

## S4 method for signature 'rf'
markers(object)

## S4 method for signature 'lg'
markers(object)

## S4 method for signature 'hetData'
markers(object)
```

Arguments

object	The mpcross object from which to extract the marker names

Value

The names of the genetic markers.

---

mpcross *Create object of class mpcross*

Description

Create object of class mpcross

Usage

```
mpcross(
  founders,
  finals,
  pedigree,
```

```
  hetData = infiniteSelfing,
  fixCodingErrors = FALSE
)
```

Arguments

`founders`	The genetic data for the founding lines of the population, represented as an integer matrix.
`finals`	The genetic data for the final lines of the population, represented as an integer matrix.
`pedigree`	An object of class `pedigree`
`hetData`	Information about how marker heterozygotes have been encoded. Can be an object of class `hetData`, or a function generating such an object from the previous three inputs.
`fixCodingErrors`	Should we automatically fix data errors, by changing invalid values to missing?

Details

This function constructs an object of class `mpcross` representing a multi-parent population. It takes in genetic data about the founding lines and final population line, a pedigree, and information about how marker heterozygotes have been encoded.

Parameter `founders` is the genetic data about the founding lines of the population. It must be an integer matrix, with rows representing genetic lines, and columns representing genetic markers. Parameter `finals` is a similar matrix, representing data for the final genetic lines in the population.

Parameter pedigree stores information about how the final lines in the population were generated from the founding lines.

Parameter hetDat must be an object of class `hetData` containing information about how marker heterozygotes have been encoded, OR a function which generates such an object. The function must take as arguments `founders`,`finals` and `pedigree`. See `infiniteSelfing` for an example of such a function.

Value

An object of class `mpcross`, constructed from the arguments.

---

mpcross-class *A collection of multi-parent populations without a genetic map*

Description

A collection of multi-parent populations without a genetic map

Details

An object of class mpcross contains data about one or more multi-parent populations, without a genetic map. As there is no genetic map, there is no information about IBD imputed genotypes or IBD genotype probabilities. There is also no information about estimated recombination fractions.

A mpcross object must contain (at a minimum) genetic data about the founding lines of the population, genetic lines about the final lines of the population, a pedigree with information about how the final lines were generated from the founding lines, and information about how heterozygotes have been encoded. See geneticData-class for further information. See mpcross for the constructor function.

Slots

geneticData A list of objects of class geneticData-class, each representing a population.

---

mpcrossMapped *Create object of class mpcrossMapped*

Description

Create object of class mpcrossMapped

Usage

```
mpcrossMapped(cross, map, rf = NULL)
```

Arguments

cross	An object of class mpcross
map	A genetic map, formatted as in the qtl package.
rf	Optional recombination fraction data. Leave as NULL if there is no such data.

Details

This function constructs an object of class mpcrossMapped representing a multi-parent population with a map. It takes in an object of class mpcross, a genetic map, and optional recombination fraction data.

Value

An object of class mpcrossMapped, constructed from the arguments.

mpcrossMapped-class *A collection of multi-parent populations with a genetic map*

Description

A collection of multi-parent populations with a genetic map

Details

An object of class `mpcrossMapped` contains genetic data for one or more populations, and a genetic map. It might also contain data about the recombination fractions between markers (or it might not).

Slots

`map` The genetic map for all populations

`rf` The recombination fraction data (which might be NULL).

`geneticData` A list of objects of class `geneticData`, each representing a population.

mpcrossRF-class *A collection of multi-parent populations with recombination fraction estimates*

Description

A collection of multi-parent populations with recombination fraction estimates

Details

An object of class `mpcrossRF` contains data about one or more multi-parent populations, without a genetic map, but with recombination fraction estimates. As there is no genetic map, there is no information about IBD imputed genotypes or IBD genotype probabilities.

Slots

`geneticData` A list of objects of class `geneticData`, each representing a population.

`rf` Estimates of recombination fractions between every pair of genetic markers.

`multiparentSNP` *Convert all markers to SNP markers*

Description

Convert all markers in an object with fully informative markers to SNP markers

Usage

```
multiparentSNP(keepHets)
```

Arguments

`keepHets`	Should heterozygotes for the SNP marker be kept?

Details

When initially generated, objects of class `mpcross` have markers that are fully informative - Every founder carries a different allele, and all marker heterozygotes are distinguishable. This function can be used to convert a simulated object to one with SNP markers. The resulting markers have two alleles, and the marker heterozygote might or might be observable.

Value

An object of internal type `multiparentSNP`, which can be combined with an object of class `mpcross` using the addition operator.

`nFounders` *Number of genotyped markers Return the number of genotyped markers in an object. If an* `mpcross` *object contains multiple experiments, one number is returned per experiment.*

Description

Number of genotyped markers

Return the number of genotyped markers in an object.

If an `mpcross` object contains multiple experiments, one number is returned per experiment.

Usage

```
nFounders(object)

## S4 method for signature 'detailedPedigree'
nFounders(object)

## S4 method for signature 'pedigree'
nFounders(object)

## S4 method for signature 'mpcross'
nFounders(object)

## S4 method for signature 'geneticData'
nFounders(object)
```

Arguments

object	The mpcross object from which to extract the number of founders

Value

The number of founding lines in the population, or a list of numbers in the case of multiple experiments contained in a single object.

---

nLines *Number of genotyped lines Return the number of genotyped lines in an object. This includes only the number of final lines genotyped in the population, and does not include the founding lines. If an* mpcross *object contains multiple experiments, one number is returned per experiment.*

Description

Number of genotyped lines

Return the number of genotyped lines in an object.

This includes only the number of final lines genotyped in the population, and does not include the founding lines. If an mpcross object contains multiple experiments, one number is returned per experiment.

Usage

```
nLines(object)

## S4 method for signature 'mpcross'
nLines(object)

## S4 method for signature 'geneticData'
nLines(object)
```

Arguments

object	The mpcross object from which to extract the number of genotyped lines.

Value

The number of genetic lines in the population, or a list of numbers in the case of multiple experiments contained in a single object.

---

nMarkers *Number of genotyped markers Return the number of genotyped markers in an object. If an* mpcross *object contains multiple experiments, all experiments are required to have the same markers. So only one number is returned, in all cases.*

Description

Number of genotyped markers

Return the number of genotyped markers in an object.

If an mpcross object contains multiple experiments, all experiments are required to have the same markers. So only one number is returned, in all cases.

Usage

```
nMarkers(object)

## S4 method for signature 'mpcross'
nMarkers(object)

## S4 method for signature 'geneticData'
nMarkers(object)
```

Arguments

object	The mpcross object from which to extract the marker names

Value

The number of markers in an object of class mpcross.

normalPhenotype *Simulate normally distributed phenotype*

Description

Add a normally distributed phenotype

Usage

```
normalPhenotype(means, standardDeviations, phenotypeName, marker)
```

Arguments

means	The means of the phenotype for all the different founder alleles
standardDeviations	The standard deviations of the phenotype for all the different founder alleles
phenotypeName	The name of the new phenotype
marker	The name of the marker which controls this phenotype

Details

Add a normally distributed phenotype to a given populations

Value

An object of class `normalPhenotype` representing the phenotype.

omp_set_num_threads *Get or set number of threads for OpenMP*

Description

Get or set the number of threads for OpenMP

Usage

```
omp_set_num_threads(num)

omp_get_num_threads()
```

Arguments

num	New number of threads for OpenMP

Details

Some functions in mpMap2 are parallelised. Depending on the number of cores available, and the type of workload, it may be advantageous to turn parallelisation on or off, by setting the number of OpenMP threads appropriately. Setting the number of threads to 1 turns parallelisation off

In particular, for small examples on a computer with a large number of threads, parallelisation may result in a huge decrease in performance.

This function returns an error if the package was not compiled with OpenMP.

Value

None

The number of threads for OpenMP

`orderCross` *Order markers Order markers within linkage groups using simulated annealing*

Description

This function orders markers within linkage groups using a simulated annealing heuristic. The underlying implementation is a C++ reimplementation of the fortran code `arsa.f` from the `seriation` package. The reimplementation allows for multithreading, and is therefore much faster. It also fixes a couple of bugs in the original code.

Parameters `cool` and `tmin` are standard simulated annealing parameters, and decreasing `cool` increases the amount of computation effort. Parameter `nReps` gives the number of independent replications of the simulated annealing algorithm to be used. The result of the best replication is then chosen.

Parameter `maxMove` gives the maximum number of positions by which to shift a marker, as part of a step within the simulated annealing algorithm. The computational effort of determining whether a proposed move of a particular marker should be accepted, depends on the number of positions by which it is moved. So if the ordering is already approximately correct at the start of the algorithm, proposals that move markers by large distances are expensive, and also unneccessary. These types of proposed changes to the ordering can be avoided by setting `maxMove` to some positive value, maybe one tenth of the number of markers.

Parameter `effortMultiplier` simply increases or decreases the amount of computational effort. A value of 0.5 requires half as much effort, a value of 1.0 uses the default amount of effort, and a value of 2.0 requires twice as much computational effort.

Parameter `randomStart` controls the starting point of each replication of the algorithm. If this parameter is TRUE, then every replication starts form an independent random ordering. If this parameter is FALSE, then every replication starts from the marker ordering given in the input object.

Usage

```
orderCross(
  mpcrossLG,
  cool = 0.5,
  tmin = 0.1,
  nReps = 1,
  maxMove = 0,
  effortMultiplier = 1,
  randomStart = TRUE,
  verbose = FALSE
)
```

Arguments

mpcrossLG	An object of class mpcrossLG, containing genetic data and linkage groups.
cool	Rate of cooling
tmin	Minimum temperature
nReps	Number of independent replications of the simulated annealing algorithm
maxMove	Maximum number of positions by which to shift a single marker, as part of the simulated annealing. A value of zero indicates no limit.
effortMultiplier	Multiplier for the amount of computational effort
randomStart	If TRUE, start from the current ordering
verbose	If TRUE, generate more detailed output

Value

An object of class mpcrossLG, identical to the input except with the markers rearranged.

---

pedigree *Create a pedigree object*

Description

Create a pedigree object

Usage

```
pedigree(lineNames, mother, father, selfing, warnImproperFunnels = TRUE)
```

Arguments

`lineNames`	The names of the genetic lines
`mother`	The index of the maternal line
`father`	The index of the paternal line
`selfing`	Should the number of generations of selfing be taken from the pedigree (`"finite"`), or should selfing be assumed to be infinite (`"infinite"`)?
`warnImproperFunnels`	Should a warning be generated in subsequent computations using this pedigree, if there are lines which do not contain all founding lines as ancestors?

Details

This function creates a pedigree object from parts. All lines are assumed to have an index, starting at 1 for the first line. Values at index of the various inputs 1 all relate to the first line, values at index 2 all relate to the second line, etc.

Input `lineNames` assigns a name to every line. Input `mother` gives the index of a mother line, where a value of 0 indicates that a line is a founder of the population (and therefore inbred). Input `father` gives the index of a father line, where a value of 0 indicates that a line is a founder of the population (and therefore inbred). Input `selfing` must be `"finite"` or `"infinite"`. A value of infinite means that the number of generations of selfing for this pedigree will be assumed to be infinite. A value of `"finite"` means that the number of generations of selfing will be computed from the pedigree, for every line.

Value

An object of class `pedigree` representing the inputs.

`pedigree-class` *Pedigree class*

Description

This class describes a pedigree for an experimental design. Although package mpMap2 only allows for the analysis of pedigrees corresponding to multi-parent crosses, this pedigree class can describe arbitrary experimental designs.

Slots

`mother` The index within the pedigree of the mother of this individual

`father` The index within the pedigree of the father of this individual

`lineName` The name of this individual

`selfing` A value indicating whether analysis of an experiment using this pedigree should assume infinite generations of selfing. A value of `"infinite"` indicates infinite generations of selfing, and a value of `"finite"` indicates finite generations of selfing.

`warnImproperFunnels` A value indicating whether to warn the user about funnels with repeated founders.

See Also

pedigree-class, simulateMPCross, detailedPedigree-class, rilPedigree, f2Pedigree, fourParentPedigreeRando
fourParentPedigreeSingleFunnel, eightParentPedigreeRandomFunnels, eightParentPedigreeSingleFunnel,
sixteenParentPedigreeRandomFunnels

---

pedigreeGraph-class *Graph for a pedigree*

Description

Graph for a pedigree

Details

This class contains the directed graph corresponding to a pedigree, and data for laying out the graph on a plane.

Slots

`graph` An object of class igraph.

`layout` A matrix where each row gives the position of a graph vertex in the plane.

---

pedigreeToGraph *Convert pedigree to a graph*

Description

Convert pedigree to a graph

Usage

```
pedigreeToGraph(pedigree)
```

Arguments

`pedigree`	The pedigree to convert into a graph

Details

It is often useful for visualisation purposes to generate the pedigree graph. In this graph, every genetic line is a vertex in a graph, and there is an edge from every parent to all the offspring. This function generates the graph, and lays the graph out in the plane in a way that tends to make the structure of the graph as clear as possible.

Value

An object of class `pedigreeGraph`, containing the graph and a planar layout for the graph.

plot,addExtraMarkersStatistics,ANY-method

Plot chi-squared statistics for independence

Description

Plot the chi-squared statistics for independence, used to map a new marker to an existing genetic map

Usage

```
## S4 method for signature 'addExtraMarkersStatistics,ANY'
plot(x, y, ...)
```

Arguments

x	Object of class addExtraMarkersStatistics containing test-statistic values.
y	Unused
...	Unused

Details

This function plots a trace of the chi-squared test-statistics used to map a new genetic marker to an existing genetic map. This can be useful to, for example, see if a single polymorphism is present at multiple points on the genome.

Value

A ggplot object suitable for display.

plot,mpcross,ANY-method

Plot methods

Description

There are multiple meaningful ways to plot some mpMap2 objects. Please use plotProbabilities or plotMosaic instead.

Usage

```
## S4 method for signature 'mpcross,ANY'
plot(x, y, ...)

## S4 method for signature 'geneticData,ANY'
plot(x, y, ...)

## S4 method for signature 'probabilities,ANY'
plot(x, y, ...)

## S4 method for signature 'imputed,ANY'
plot(x, y, ...)
```

Arguments

x	Unused
y	Unused
...	Unused

Details

There are multiple meaningful ways to plot some mpMap2 objects. In these cases the plot function is implemented but returns an error. Please use `plotProbabilities` or `plotMosaic` instead.

Value

None

---

plot,pedigreeGraph,ANY-method

Plot the graph of a pedigree

Description

Plot the graph of a pedigree

Usage

```
## S4 method for signature 'pedigreeGraph,ANY'
plot(x, y, ...)
```

Arguments

x	pedigree graph to plot
y	unused
...	Other options to plot.igraph

Details

Plot the graph of a pedigree, after the graph has been generated by pedigreeToGraph

Value

None

See Also

pedigreeToGraph

`plotMosaic` *Plot estimated genetic composition of lines*

Description

Plot estimated genetic composition of lines

Usage

```
plotMosaic(inputObject, chromosomes, positions, lines, ...)
```

Arguments

`inputObject`	An object of class `mpcrossMapped` containing imputed IBD genotypes
`chromosomes`	Chromosomes to plot
`positions`	Genetic positions to plot
`lines`	Genetic lines to plot
`...`	Extra inputs to `heatmap_2`

Details

This function produces a heatmap showing the genetic composition of lines, as measured by the imputed IBD genotypes. Rows correspond to genetic lines, columns correspond to genetic positions, and colours indicate founder alleles. All heterozygotes are marked in the same colour, otherwise there are generally too many colours to be useful.

Value

None

`plotProbabilities` *Plot genetic composition across the genome*

Description

Plot genetic composition across the genome

Usage

```
plotProbabilities(inputObject, positions, alleles, chromosomes)
```

Arguments

`inputObject`	An object of class `mpcrossMapped` containing IBD genotype probabilities
`positions`	The genetic positions at which to plot the composition
`alleles`	The founder alleles which we are interested in.
`chromosomes`	The chromosomes of to plot the composition.

Details

Plot genetic composition of a population, across the genome. Composition is determined by using the IBD genotype probabilities, as computed by `computeGenotypeProbabilities`. The plot is produced by taking the average IBD genotype probability, for each founder allele and each genotpe position. Deviations from the expected proprotions (determined by the experimental design) may indicate non-standard genetic inheritance or selective pressure.

Value

A ggplot object, suitable for display.

Examples

```
data(simulatedFourParentData)
part1 <- subset(simulatedFourParentData, lines =
names(which(finals(simulatedFourParentData)[, 50] == 1)))
part2 <- subset(simulatedFourParentData, lines =
names(which(finals(simulatedFourParentData)[, 50] != 1)))
distorted <- subset(part1, lines = sample(nLines(part1), 100)) + part2
distortedMapped <- mpcrossMapped(distorted, map = simulatedFourParentMap)
probabilities <- computeGenotypeProbabilities(distortedMapped)
#Here the composition of the population reflects the fact that we have less of founder 1 than
#    expected, at a specific point on the genome
plotProbabilities(probabilities)
#Go back to the undistorted data
undistortedMapped <- mpcrossMapped(simulatedFourParentData, map = simulatedFourParentMap)
probabilities <- computeGenotypeProbabilities(distortedMapped)
#Here the composition of the population reflects the expected inheritance; the trace
#    corresponding to every founder is flat
plotProbabilities(probabilities)
```

probabilities-class *Identity-by-descent genotype probabilities*

Description

Identity-by-descent genotype probabilities

Arguments

data	An integer matrix containing the IBD genotype probabilities. Rows correspond to combinations of genetic lines and founder lines, and columns correspond to genetic positions.
key	A matrix identifying how pairs of founder alleles are mapped to rows of the data slot.
map	The map of positions at which IBD genotype probabilities are computed.

Details

This object contains the identify-by-descent genotype probabilities, as computed by computeGenotypeProbabilities. The slot data is a numeric matrix containing the actual computations, where columns correspond to genetic positions.

Describing the rows of the data matrix is more complicated. The slot key is a matrix containing three columns, the first two being founder alleles, and the third being an encoding of that combination. If k is the number of rows in key, then the first k rows of the data matrix correspond to the first genetic line in the population. Specifically, the first row corresponds to genotype probabilities for the first line, for the combination of founder alleles encoding as 1. The second corresponds to genotype probabilities for the first line, for the combination encoded as 2, etc.

probabilityData *Get IBD probability data*

Description

Get the identity-by-descent probability data from an mpcross object.

Usage

```
probabilityData(object, ...)

## S4 method for signature 'geneticData'
probabilityData(object, ...)

## S4 method for signature 'mpcrossMapped'
probabilityData(object, ...)
```

Arguments

object	The mpcross object from which to extract the probability data.
...	Additional options. Only design is supported, and gets the probability data for only a single experiment.

Details

mpMap2 stores IBD probabilities in a matrix where the number of rows is the number of alleles times the number of genetic lines, and the columns are the positions at which probabilities are calculated. In the example below, the row names are L115 -L1,L115 -L2,L115 -L3,L115 -L4,L120 -L1, etc, and the column names are DXM1,DXM2,DXM3, etc. So, for example, for a population generated from four founders and assumed to be totally inbred, the first four values in the first column are the probabilities that genetic line 1 carries alleles from specific founders, at a specific position. The first four columns give the probabilities for genetic line 2 at the next position, etc.

This can be an inconvenient layout for some operations. This function returns a matrix where the alleles appear as part of the columns, rather than the rows. For example, after applying this function to the given example, the first four values in the first row will be the probabilities that genetic line 1 carries alleles from specific founders, at a specific position.

Value

A numeric matrix containing the IBD probabliity data, or a list of such matrices in the case of multiple experiments within a single object.

Examples

```
data(simulatedFourParentData)
crossSNP <- simulatedFourParentData + multiparentSNP(keepHets = FALSE)
mapped <- mpcrossMapped(crossSNP, map = simulatedFourParentMap)
probabilities <- computeGenotypeProbabilities(mapped, error = 0.05)
probabilityData <- probabilityData(probabilities)
probabilityData[1:5, 1:5]
```

---

redact *Redact sensitive information This function redacts possibly sensitive information from objects, resulting in an object that is safe to publish.*

Description

Sensitive information includes names of genetic lines (both founding lines and final population lines) and marker names. All actual data (marker genotypes, imputed IBD genotypes, IBD probabilities, etc) are preserved.

Usage

```
redact(object)

## S4 method for signature 'mpcross'
redact(object)

## S4 method for signature 'mpcrossRF'
redact(object)

## S4 method for signature 'mpcrossLG'
redact(object)

## S4 method for signature 'mpcrossMapped'
redact(object)

## S4 method for signature 'geneticData'
redact(object)
```

Arguments

object	The object of class mpcross to redact.

Value

An object of class mpcross, with identifying information removed.

---

removeHets *Remove heterozygotes*

Description

Remove all heterozygotes from dataset

Usage

```
removeHets()
```

Details

This function can be used to remove all heterozygotes from an mpcross object. Information about how pairs of different marker alleles are encoded as genotypes is discarded, and all observations of heterozygotes will be marked as NA. Any information calculated based on the genetic data (imputed IBD genotypes, IBD probabilities) will be discarded.

Value

An object of internal class removeHets, which can be combined with an object of class mpcross using the addition operator.

Examples

```
pedigree <- eightParentPedigreeImproperFunnels(initialPopulationSize = 10,
     selfingGenerations = 1, nSeeds = 1)
#Generate map
map <- qtl::sim.map()
#Simulate data
cross <- simulateMPCross(map = map, pedigree = pedigree, mapFunction = haldane)
finals(cross)[1:5, 1:5]
hetData(cross)[[1]]
cross <- cross + removeHets()
finals(cross)[1:5, 1:5]
hetData(cross)[[1]]
```

---

reverseChromosomes *Reverse the order of the specified chromosomes*

Description

Create a new object, with the specified chromosomes reversed

Usage

```
reverseChromosomes(mpcrossMapped, chromosomes)
```

Arguments

mpcrossMapped	The initial object, for which we want to reverse some of the chromosomes
chromosomes	The names of the chromosomes to reverse

Details

Create a new object, with the specified chromosomes reversed

Value

An object of class mpcrossMapped, with certain chromosomes reversed.

Examples

```
map <- qtl::sim.map()
pedigree <- f2Pedigree(1000)
cross <- simulateMPCross(map = map, pedigree = pedigree, mapFunction = haldane, seed = 1)
mappedCross <- mpcrossMapped(cross = cross, map = map)
reversedX <- reverseChromosomes(mappedCross, "X")
reversedX@map[["X"]]
mappedCross@map[["X"]]
```

rilPedigree *Generate a two-parent RIL pedigree which starts from inbred founders*

Description

Generate a two-parent RIL pedigree which starts from inbred founders

Usage

```
rilPedigree(populationSize, selfingGenerations)
```

Arguments

populationSize	The size of the generated population
selfingGenerations	Number of generations of selfing. Specifying one generation leads to an F2 design.

Value

An object of class `detailedPedigree` representing the experimental design, suitable for simulation using simulateMPCross.

selfing<- *Get or set a pedigree to have finite or infinite generations of selfing*

Description

Get or set a pedigree to have finite or infinite generations of selfing

Usage

```
selfing(object) <- value

selfing(object)

## S4 method for signature 'pedigree'
selfing(object)

## S4 replacement method for signature 'detailedPedigree'
selfing(object) <- value

## S4 replacement method for signature 'pedigree'
selfing(object) <- value
```

Arguments

object	The pedigree object for which to get or set the generations of selfing, as finite or infinite.
value	The new value

Details

A pedigree object contains details about the genetic relationships between individuals in a population. Many experiments will include a finite number of generations of inbreeding by selfing, and this information will also be contained in the pedigree. However, when it comes time to actually analyse the poulation, it can be sensible to assume that an infinite number of generations of selfing have actually been performed, as this is computationally quicker.

This extra information about whether to assume infinite generations of selfing, or the finite number of generations given in the pedigree, is stored in an extra slot, which must have value `"finite"` or `"infinite"`. If `"finite"` is specified, then in subsequent analysis (e.g. computation of IBD genotypes or probabilities) the number of generations of selfing for each line is taken from the pedigree.

Value

Dimensions of selfing, either `"finite"` or `"infinite"`.

Examples

```
pedigree <- eightParentPedigreeImproperFunnels(initialPopulationSize = 10,
     selfingGenerations = 0, nSeeds = 1)
selfing(pedigree)
selfing(pedigree) <- "finite"
```

---

simulatedFourParentData

Simulated data from a four-parent population.

Description

Simulated data from a four-parent population. Used in the examples given in the documentation.

Examples

```
set.seed(1)
#This data was generated by the following script
pedigree <- fourParentPedigreeRandomFunnels(initialPopulationSize = 1000,
     selfingGenerations = 6, intercrossingGenerations = 0)
#Assume infinite generations of selfing in subsequent analysis
selfing(pedigree) <- "infinite"
#Generate random map
simulatedFourParentMap <- qtl::sim.map(len = 100, n.mar = 101, anchor.tel = TRUE,
```

```
include.x = FALSE)
#Simulate data
simulatedFourParentData <- simulateMPCross(map = simulatedFourParentMap, pedigree = pedigree,
mapFunction = haldane, seed = 1L)
```

`simulateMPCross` *Simulate data from multi-parent designs*

Description

Data is simulated according to a pedigree, map and QTL model

Usage

```
simulateMPCross(map, pedigree, mapFunction, seed)
```

Arguments

`map`	Linkage map with which to generate data. See `sim.map`
`pedigree`	Pedigree for a multi-parent cross.
`mapFunction`	Map function used to convert distances to recombination fractions
`seed`	Random seed to use.

Value

Object of class `mpcross`.

`sixteenParentPedigreeRandomFunnels`
Generate a sixteen-parent pedigree

Description

Generate a sixteen-parent pedigree starting from inbred founders, using a random funnel

Usage

```
sixteenParentPedigreeRandomFunnels(
  initialPopulationSize,
  selfingGenerations,
  nSeeds = 1L,
  intercrossingGenerations
)
```

Arguments

initialPopulationSize	The number of F1 lines generated
selfingGenerations	The number of selfing generations at the end of the pedigree
nSeeds	The number of progeny taken from each intercrossing line, or from each F1 if no intercrossing is specified. These lines are then selfed according to selfingGenerations
intercrossingGenerations	The number of generations of random mating performed from the F1 generation. Population size is maintained at that specified by initialPopulationSize

Value

An object of class `detailedPedigree` representing the experimental design, suitable for simulation using simulateMPCross.

See Also

`eightParentPedigreeSingleFunnel`, `fourParentPedigreeSingleFunnel`, `fourParentPedigreeRandomFunnels`, `twoParentPedigree`

---

stripPedigree *Strip pedigree of unneccessary lines*

Description

Strip pedigree of lines that make no genetic contribution to the specified set of lines.

Usage

```
stripPedigree(pedigree, finalLines)
```

Arguments

pedigree	The initial pedigree, which may contain some unneccessary extra genetic lines
finalLines	The list of lines of interest. Lines in the pedigree which do not make a genetic contribution to the lines in `finalLines` will be removed.

Details

Pedigrees for structured experiments can be messy. Often they include lines that make no genetic contribution to the lines that were finally genotyped. When it comes to visualising the structure of the pedigree of the final population, these unneccessary extra lines can make it difficult to see the structure. This function takes in a pedigree and a list of genetic lines, and returns a subpedigree that contains only those lines that make a genetic contribution to the named lines.

This function relies on the use of the Boost C++ libraries, and may not be available in every distributed version of mpMap2. If this function is unavailable, the function will return `NULL`.

Value

An object of class `detailedPedigree` representing the experimental design, suitable for simulation using simulateMPCross.

`subset,imputed-method` *Subset data*

Description

Subset data objects by line names, chromosomes, linkage groups, markers or positions

Usage

```
## S4 method for signature 'imputed'
subset(x, ...)

## S4 method for signature 'probabilities'
subset(x, ...)

## S4 method for signature 'mpcross'
subset(x, ...)

## S4 method for signature 'mpcrossMapped'
subset(x, ...)

## S4 method for signature 'mpcrossRF'
subset(x, ...)

## S4 method for signature 'mpcrossLG'
subset(x, ...)

## S4 method for signature 'lg'
subset(x, ...)

## S4 method for signature 'geneticData'
subset(x, ...)

## S4 method for signature 'hetData'
subset(x, ...)

## S4 method for signature 'rf'
subset(x, ...)

## S4 method for signature 'rawSymmetricMatrix'
subset(x, ...)
```

Arguments

x	The object to be subset
...	A method to use to subset (markers, lines, positions or chromosomes), and values for that method.

Details

mpMap2 objects can be subset in a number of different ways, depending on the particular class of the object that is contained.

Subsetting by "lines" subsets by the genetic lines in the final population. Line names or line indices can be used, although line names should be preferred. Any information about recombination fractions will be discarded. Subsetting by "chromosomes" keeps only certain chromosomes, and requires that the object have a genetic map. Subsetting by "markers" keeps only certain genetic markers. Data about imputed IBD genotypes and IBD genotype probabilities is discarded. Subsetting by "positions" only subsets the imputed IBD genotypes and IBD probability data, and does not subset the underlying markers. Subestting by "groups" retains only certain linkage groups.

An object of class mpcross can be subset by genetic lines or markers.

Objects of classes mpcrossLG or mpcrossRF can be subset by genetic lines, markers or linkage groups.

An object of class mpcrossMapped can be subset by genetic lines, markers or chromosomes.

The remainder of the subsetting methods are not expected to be called directly by the user. They subset internal components, and are used internally by the top-level methods.

Value

A subsetted object, of the same type as the input.

---

testDistortion *Test for distortion using IBD genotype probabilities*

Description

Test for distortion using IBD genotype probabilities

Usage

```
testDistortion(object)
```

Arguments

object	An object of class mpcrossMapped which contains imputed IBD genotype data

Details

In real experiments, genetic inheritance may not follow the expected model. This function tests for deviations from expected inheritance by using the genetic composition of the population at individual positions, as measured by the IBD genotype probabilities.

At a particular point, the mean for each founder allele of the IBD genotype probabilities for each founder allele are summed across the population. The average is taken, and this is then compared with the proportion expected to be inherited from that founder, under standard models of genetic inheritance.

The result is a matrix containing p-values, test-statistic values, and the L1 and L2 distances between the observed genetic proportions, and the expected genetic proportions.

Value

A data.frame containing p-values and test-statistic values for each position at which there is IBD genotype probability data.

---

toMpMap *Convert to mpMap format*

Description

Convert to the format used by the original mpMap package.

Usage

```
toMpMap(mpcross)
```

Arguments

`mpcross`	The object of class `mpcross` to convert.

Details

Converts an `mpcross` object to the format used by the original mpMap, the predecessor of this package. It is unlikely that this function will ever need to be used.

Value

An object with structure compatible with the older mpMap package.

transposeProbabilities *Transpose IBD probabilities*

Description

Transpose the IBD probabilities matrix, so that the different alleles or founders appear on the columns, rather than the rows

Usage

```
transposeProbabilities(inputObject)
```

Arguments

inputObject	The mpcross object containing the probability data.

Details

mpMap2 stores IBD probabilities in a matrix where the number of rows is the number of alleles times the number of genetic lines, and the columns are the positions at which probabilities are calculated. In the example below, the row names are L115 -L1,L115 -L2,L115 -L3,L115 -L4,L120 -L1, etc, and the column names are DXM1,DXM2,DXM3, etc. So, for example, for a population generated from four founders and assumed to be totally inbred, the first four values in the first column are the probabilities that genetic line 1 carries alleles from specific founders, at a specific position. The first four columns give the probabilities for genetic line 2 at the next position, etc.

This can be an inconvenient layout for some operations. This function returns a matrix where the alleles appear as part of the columns, rather than the rows. For example, after applying this function to the given example, the first four values in the first row will be the probabilities that genetic line 1 carries alleles from specific founders, at a specific position.

Value

A numeric matrix containing IBD probability data.

Examples

```
data(simulatedFourParentData)
crossSNP <- simulatedFourParentData + multiparentSNP(keepHets = FALSE)
mapped <- mpcrossMapped(crossSNP, map = simulatedFourParentMap)
probabilities <- computeGenotypeProbabilities(mapped, error = 0.05)
probabilityData <- probabilityData(probabilities)
probabilityData[1:5, 1:5]
transposeProbabilities(probabilities)[1:5,1:5]
```

`twoParentPedigree` *Generate a two-parent pedigree which starts from inbred founders*

Description

Generate a two-parent pedigree starting from inbred founders

Usage

```
twoParentPedigree(
  initialPopulationSize,
  selfingGenerations,
  nSeeds = 1L,
  intercrossingGenerations
)
```

Arguments

`initialPopulationSize`	The number of F1 lines generated
`selfingGenerations`	The number of selfing generations at the end of the pedigree
`nSeeds`	The number of progeny taken from each intercrossing line, or from each F1 if no intercrossing is specified. These lines are then selfed according to selfingGenerations
`intercrossingGenerations`	The number of generations of random mating performed from the F1 generation. Population size is maintained at that specified by initialPopulationSize

Value

An object of class `detailedPedigree` representing the experimental design, suitable for simulation using simulateMPCross.

Examples

```
plotWOptions <- function(graph)
plot(graph, vertex.size = 8, vertex.label.cex=0.6, edge.arrow.size=0.01, edge.width=0.2)
#F2 design
pedigree <- twoParentPedigree(initialPopulationSize = 10, selfingGenerations = 1,
intercrossingGenerations = 0, nSeeds = 1)
graph <- pedigreeToGraph(pedigree)
plotWOptions(graph)

#An equivalent F2 design (if the founders really are inbred)
pedigree <- twoParentPedigree(initialPopulationSize = 10, selfingGenerations = 0,
intercrossingGenerations = 1, nSeeds = 0)
graph <- pedigreeToGraph(pedigree)
```

```
plotWOptions(graph)

#Another equivalent F2 design (if the founders really are inbred)
pedigree <- twoParentPedigree(initialPopulationSize = 1, selfingGenerations = 1,
intercrossingGenerations = 0, nSeeds=10)
graph <- pedigreeToGraph(pedigree)
plotWOptions(graph)

#A RIL design (10 generations of inbreeding)
pedigree <- twoParentPedigree(initialPopulationSize = 10, selfingGenerations = 10,
intercrossingGenerations = 0, nSeeds = 1)
graph <- pedigreeToGraph(pedigree)
plotWOptions(graph)

#Another RIL design (10 generations of inbreeding)
pedigree <- twoParentPedigree(initialPopulationSize = 1, selfingGenerations = 10,
intercrossingGenerations = 0, nSeeds = 10)
graph <- pedigreeToGraph(pedigree)
plotWOptions(graph)
#One generation of mixing followed by 10 generations of inbreeding
pedigree <- twoParentPedigree(initialPopulationSize = 10, selfingGenerations = 10,
intercrossingGenerations = 1, nSeeds = 1)
graph <- pedigreeToGraph(pedigree)
plotWOptions(graph)

#Two generations of mixing and no inbreeding
pedigree <- twoParentPedigree(initialPopulationSize = 10, selfingGenerations = 0,
intercrossingGenerations = 2, nSeeds = 0)
graph <- pedigreeToGraph(pedigree)
plotWOptions(graph)

#One generation of mixing, and then two selfed lines are generated (10 generations of selfing)
pedigree <- twoParentPedigree(initialPopulationSize = 10, selfingGenerations = 10,
intercrossingGenerations = 1, nSeeds = 2)
graph <- pedigreeToGraph(pedigree)
plotWOptions(graph)
```

---

wsnp_Ku_rep_c103074_89904851

Raw genotyping data for marker wsnp_Ku_rep_c103074_89904851

Description

Raw genotyping data for marker wsnp_Ku_rep_c103074_89904851

Author(s)

Alex Whan, Matthew Morell, Rohan Shah, Colin Cavanagh This dataset contains the raw genotyping data for marker wsnp_Ku_rep_c103074_89904851. This marker is interesting, because it can be mapped to both chromosome 1B (four alleles) and 1D (two alleles).

Examples

```
data(eightParentSubsetMap)
data(wsnp_Ku_rep_c103074_89904851)
data(callFromMapExampleLocalisationStatistics)
called <- callFromMap(rawData = as.matrix(wsnp_Ku_rep_c103074_89904851), existingImputations =
    eightParentSubsetMap, useOnlyExtraImputationPoints = TRUE, tDistributionPValue = 0.8,
   thresholdChromosomes = 80, existingLocalisationStatistics = existingLocalisationStatistics)
library(ggplot2)
library(gridExtra)
plotData <- wsnp_Ku_rep_c103074_89904851
plotData$genotype1B <- factor(called$classificationsPerPosition$Chr1BLoc31$finals)
plotData$imputed1B <- factor(imputationData(eightParentSubsetMap)[, "Chr1BLoc31"])
plotData$genotype1D <- factor(called$classificationsPerPosition$Chr1DLoc16$finals)
plotData$imputed1D <- factor(imputationData(eightParentSubsetMap)[, "Chr1DLoc16"])

plotImputations1B <- ggplot(plotData, mapping = aes(x = theta, y = r, color = imputed1B)) +
    geom_point() + theme_bw() + ggtitle("Imputed genotype, 1B") +
    guides(color=guide_legend(title="IBD genotype"))

called1B <- ggplot(plotData, mapping = aes(x = theta, y = r, color = genotype1B)) +
    geom_point() + theme_bw() + ggtitle("Called genotype, 1B") +
    guides(color=guide_legend(title="Called cluster")) + scale_color_manual(values =
    c("black", RColorBrewer::brewer.pal(n = 4, name = "Set1")))

plotImputations1D <- ggplot(plotData, mapping = aes(x = theta, y = r, color = imputed1D)) +
    geom_point() + theme_bw() + ggtitle("Imputed genotype, 1D") +
    guides(color=guide_legend(title="IBD genotype"))

called1D <- ggplot(plotData, mapping = aes(x = theta, y = r, color = genotype1D)) +
    geom_point() + theme_bw() + ggtitle("Called genotype, 1D") +
    guides(color=guide_legend(title="Called cluster")) +
   scale_color_manual(values = c("black",RColorBrewer::brewer.pal(n=3,name = "Set1")[1:2]))

grid.arrange(plotImputations1B, plotImputations1D, called1B, called1D)
```

[,rawSymmetricMatrix,index,index,logical-method
Internal operators for mpMap2

Description

Internal operators, used to modify mpcross objects.

Usage

```
## S4 method for signature 'rawSymmetricMatrix,index,index,logical'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'rawSymmetricMatrix,index,index,missing'
```

```
x[i, j, ..., drop = TRUE]

## S4 method for signature 'rawSymmetricMatrix,missing,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'rawSymmetricMatrix,matrix,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'geneticData,assignFounderPattern'
e1 + e2

## S4 method for signature 'mpcross,assignFounderPattern'
e1 + e2

## S4 method for signature 'mpcrossMapped,assignFounderPattern'
e1 + e2

## S4 method for signature 'geneticData,assignFounderPatternPrototype'
e1 + e2

## S4 method for signature 'mpcross,assignFounderPatternPrototype'
e1 + e2

## S4 method for signature 'mpcrossMapped,assignFounderPatternPrototype'
e1 + e2

## S4 method for signature 'geneticData,biparentalDominant'
e1 + e2

## S4 method for signature 'mpcross,biparentalDominant'
e1 + e2

## S4 method for signature 'geneticData,fixedNumberOfFounderAlleles'
e1 + e2

## S4 method for signature 'mpcross,fixedNumberOfFounderAlleles'
e1 + e2

## S4 method for signature 'geneticData,multiparentSNP'
e1 + e2

## S4 method for signature 'mpcross,multiparentSNP'
e1 + e2

## S4 method for signature 'geneticData,multiparentSNPPrototype'
e1 + e2

## S4 method for signature 'mpcross,multiparentSNPPrototype'
```

```
e1 + e2

## S4 method for signature 'mpcross,removeHets'
e1 + e2

## S4 method for signature 'geneticData,normalPhenotype'
e1 + e2

## S4 method for signature 'mpcross,normalPhenotype'
e1 + e2
```

Arguments

x	object from which to extract element(s)
i, j	indices specifying elements to extract or replace
...	Currently unused
drop	If TRUE the result is coerced to the lowest possible dimension
e1	Object one
e2	Object two

Details

These operators are used to combine objects in order to apply a function. For example, `e1 + biparentalDominant()` returns the biparental design `e1`, with all markers converted to dominant markers. Consult the documentation on the individual functions, rather than this list of operators.

Value

Various. Not for external use.

Modified version of the input object. The class of the output depends on the class of the input.

Index