Package 'inferRecom'

November 25, 2026

Title Homologous Recombination Detection in Nuclear Families

Version 1.0.0

Description This package implements a nuclear family-based algorithm to detect homologous recombination. Using genotypes in PLINK binary format, recombination events are inferred from changes in the inheritance state of informative SNPs between siblings. Closely spaced crossovers that are unlikely to represent true recombination can be filtered out using genetic maps or a SNP-count filter. Additional functions are supplied to detect runs of homozygosity and to phase haplotypes at informative SNPs and intervals that are inferred identical-by-descent. Recombination intervals and runs of homozygosity are returned as `GRanges` objects, and phased genotypes are returned as a `VariantAnnotation` `VCF` object, for use with other Bioconductor packages and genome browsers.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

VignetteBuilder knitr

biocViews Genetics, SNP, GenomeAnnotation, Sequencing, Coverage, DataImport

Depends R (>= 4.6.0), VariantAnnotation, GenomicRanges, S4Vectors

Imports methods, utils, Biostrings, IRanges, snpStats, BiocParallel, dplyr, stats, tools

Suggests testthat (>= 3.0.0), knitr, rmarkdown, BiocStyle, rtracklayer, trackViewer

BugReports https://github.com/catherinefayemahoney/inferRecom/issues

URL https://github.com/catherinefayemahoney/inferRecom

Config/testthat/edition 3

git_url https://git.bioconductor.org/packages/inferRecom

git_branch RELEASE_3_24

git_last_commit dd81292

git_last_commit_date 2026-11-23

Repository Bioconductor 3.24

Date/Publication 2026-11-24

Author Catherine Mahoney [aut, cre] (ORCID:
<https://orcid.org/0000-0003-0424-104X>),
Michael Salter-Townshend [aut] (ORCID:
<https://orcid.org/0000-0001-6232-9109>),
Denis Shields [aut] (ORCID: <https://orcid.org/0000-0003-4015-2474>),
Irish Research Council [fnd] (Grant EPSPG/2019/443)

Maintainer Catherine Mahoney <catherine.f.mahoney@gmail.com>

Contents

.addHomozygousSNPs . . . 3
.addNonInformativeSNPs . . . 3
.alignSnpData . . . 4
.alignToSnps . . . 4
.assignAlleleCodes . . . 5
.bimFromMap . . . 5
.checkCrossovers . . . 6
.checkSameChromosome . . . 6
.completeHeterozygote . . . 7
.convertToNucleotides . . . 7
.detectRohAllSamples . . . 8
.detectRohSample . . . 8
.familiesToPhase . . . 9
.familyGenotypes . . . 10
.getInformativeGenotypes . . . 10
.getOtherAllele . . . 11
.getSnpLocations . . . 11
.impliedHap1Alleles . . . 12
.inheritanceIndex . . . 12
.inheritedHaplotype . . . 13
.isHomozygous . . . 14
.matchAlleleCode . . . 14
.parentAlleleTables . . . 15
.phaseBothHeterozygous . . . 15
.phaseBothHomozygous . . . 16
.phaseChildren . . . 17
.phasedVcfHeader . . . 17
.phaseListToVcf . . . 18
.phaseNonInformativeSnp . . . 18
.phaseOneHeterozygous . . . 19
.phaseParent . . . 20
.phaseSingleFamily . . . 21
.rohResultDataFrame . . . 21
.selectSamples . . . 22
.transmittedAllele . . . 22
.undoCrossovers . . . 23
.vcfSampleData . . . 23
crossoverData . . . 24
geneticMaps-data . . . 25
hzRun . . . 26

simCEU-data 28
xoDetect 29
xoPhase 31

Index **35**

`.addHomozygousSNPs` *Add Homozygous SNPs from Other Parent*

Description

Add Homozygous SNPs from Other Parent

Usage

```
.addHomozygousSNPs(actgMatrixHom, actgMatrixHet, genotypeDf, bim, parent)
```

Arguments

`actgMatrixHom`	Matrix for parent being imputed
`actgMatrixHet`	Matrix from other parent (het SNPs)
`genotypeDf`	Original genotype data frame
`bim`	BIM data frame
`parent`	"father" or "mother"

Value

Combined data frame with het and hom SNPs

`.addNonInformativeSNPs`
Add Non-Informative SNPs to Phased Output

Description

Adds SNPs that were not informative for phasing back into the output, with alleles inferred from the parental genotypes where possible (see `.phaseBothHomozygous()` and `.phaseOneHeterozygous()`).

Usage

```
.addNonInformativeSNPs(
  phasedData,
  genoSubset,
  bim,
  fatherId,
  motherId,
  fatherCol,
  motherCol
)
```

Arguments

phasedData	Data frame with phased informative SNPs
genoSubset	Full genotype matrix for the family
bim	BIM data frame with SNP information
fatherId	Father identifier
motherId	Mother identifier
fatherCol	Column index for father in genoSubset
motherCol	Column index for mother in genoSubset

Value

Data frame with all SNPs (informative and non-informative)

.alignSnpData *Align SNP Data to Common Set*

Description

Ensures a data frame contains all specified SNPs in order, filling missing SNPs with NA values

Usage

```
.alignSnpData(snpDf, allSnps)
```

Arguments

snpDf	Data frame with SNP information (must have rsID column)
allSnps	Character vector of all SNPs to include

Value

Data frame with all SNPs aligned

.alignToSnps *Align Phased Haplotypes to SNPs*

Description

Align Phased Haplotypes to SNPs

Usage

```
.alignToSnps(phase, snps)
```

Arguments

phase	Data frame of phased haplotypes with SNP IDs as row names
snps	SNP IDs in the required order

Value

The rows of phase for snps, in that order, without row names

.assignAlleleCodes *Assign Allele Codes for One Parent*

Description

Determines which allele (1 or 2) was inherited from specified parent

Usage

```
.assignAlleleCodes(genoDf, parent, fatherCol, motherCol)
```

Arguments

genoDf	Informative genotype matrix
parent	"father" or "mother"
fatherCol	Column index for father
motherCol	Column index for mother

Value

Data frame with allele codes

.bimFromMap *SNP Information from the PLINK Map*

Description

SNP Information from the PLINK Map

Usage

```
.bimFromMap(map)
```

Arguments

map	SNP map from `read.plink()`

Value

Data frame with columns chr, snp, cm, pos, a1 and a2, in BIM file column order

.checkCrossovers *Check Crossover Input*

Description

Checks that a crossover object is a GRanges on a single chromosome, and warns if it is empty.

Usage

```
.checkCrossovers(x, name = deparse(substitute(x)))
```

Arguments

x	Crossover results from xoDetect()
name	Argument name used in messages

Value

Invisibly TRUE; stops with an error otherwise

.checkSameChromosome *Check if all chromosomes in GRanges are the same*

Description

Check if all chromosomes in GRanges are the same

Usage

```
.checkSameChromosome(gr)
```

Arguments

gr	GRanges object

Value

Logical; TRUE if all chromosomes are identical

.completeHeterozygote *Alleles of a Heterozygous Child*

Description

Alleles of a Heterozygous Child

Usage

```
.completeHeterozygote(fromParents, alleles)
```

Arguments

fromParents	Alleles received from the father and the mother, NA where unknown
alleles	Allele 1 and allele 2 of the SNP

Value

The paternal and maternal alleles, or NULL if neither is known or both are known and equal

.convertToNucleotides *Convert Allele Codes to Nucleotides*

Description

Convert Allele Codes to Nucleotides

Usage

```
.convertToNucleotides(alleleCodeDf, bim)
```

Arguments

alleleCodeDf	Data frame with allele codes (1 or 2)
bim	BIM data frame with allele information

Value

Data frame with nucleotide alleles: allele 1 or 2 from the BIM file, or "" for other codes and for SNPs that are not in the BIM file

.detectRohAllSamples *Detect ROH Across All Samples*

Description

Efficiently processes all samples to detect runs of homozygosity with optional parallel processing

Usage

```
.detectRohAllSamples(
  genoMatrix,
  locationVec,
  snpNames,
  minSnps,
  minBp,
  geneticMap = NULL,
  minCm = NULL,
  BPPARAM = SerialParam()
)
```

Arguments

genoMatrix	Matrix of genotypes (SNPs x samples)
locationVec	Physical positions of SNPs in base pairs
snpNames	SNP identifiers
minSnps	Minimum number of SNPs
minBp	Minimum physical length in base pairs
geneticMap	Data frame with genetic map, or NULL
minCm	Minimum genetic length in cM, or NULL
BPPARAM	BiocParallelParam object for parallel execution

Value

List of matrices, one per sample with detected ROH

.detectRohSample *Detect ROH for a Single Sample*

Description

Identifies runs of homozygosity in one individual

Usage

```
.detectRohSample(
  genoVec,
  locationVec,
  snpNames,
  sampleId,
  minSnps,
  minBp,
  geneticMap = NULL,
  minCm = NULL
)
```

Arguments

genoVec	Numeric vector of genotypes for one sample (0, 1, 2, or NA)
locationVec	Physical positions of SNPs in base pairs
snpNames	SNP identifiers
sampleId	Sample identifier
minSnps	Minimum number of SNPs
minBp	Minimum physical length in base pairs
geneticMap	Data frame with genetic map (chr, pos, cM), or NULL
minCm	Minimum genetic length in cM, or NULL

Value

Matrix with ROH information, or NULL if no ROH detected

.familiesToPhase *Families to Phase*

Description

Families to Phase

Usage

```
.familiesToPhase(fam, famIds)
```

Arguments

fam	FAM data frame
famIds	Character vector of family IDs, or NULL

Value

famIds if given, otherwise the IDs of all families with 5 or more members (two parents and three or more children), in file order

.familyGenotypes *Phased Genotypes for One Family*

Description

Phased Genotypes for One Family

Usage

```
.familyGenotypes(famPhase, snps, ref, alt)
```

Arguments

famPhase	DataFrame of phased haplotypes for one family, with <childID>Pat/<childID>Mat and <parentID>_1/<parentID>_2 columns
snps	Character vector of SNP IDs (output rows)
ref	Character vector of REF alleles, matching snps
alt	Character vector of ALT alleles, matching snps

Value

Character matrix of phased GT values (SNPs x family members)

.getInformativeGenotypes *Get Informative Genotypes for Phasing*

Description

Filters for informative SNPs and checks Mendelian consistency

Usage

```
.getInformativeGenotypes(genoSubset)
```

Arguments

genoSubset	Genotype matrix for one family

Value

Matrix of informative genotypes or NULL

.getOtherAllele *Get Complementary Allele*

Description

Vectorised: looks all the SNPs up in the BIM file with one `match()`.

Usage

```
.getOtherAllele(snpId, allele, bim)
```

Arguments

`snpId`	SNP identifiers
`allele`	Current alleles (A, C, G, or T), one per SNP
`bim`	BIM data frame

Value

Complementary alleles, NA for a SNP that is not in the BIM file

.getSnpLocations *Get Physical Positions for SNPs*

Description

Map SNP names to physical positions

Usage

```
.getSnpLocations(snpNames, mapData)
```

Arguments

`snpNames`	Character vector of SNP names
`mapData`	Data frame with SNP map information

Value

Numeric vector of positions

.impliedHap1Alleles *Heterozygous Parent's Haplotype 1 as Implied by Each Child*

Description

A child that inherited the parent's haplotype *k* around the SNP (see `.inheritedHaplotype()`) shows that haplotype *k* carries the allele the child received.

Usage

```
.impliedHap1Alleles(
  received,
  alleles,
  phasedData,
  hetId,
  hetSuffix,
  snpPos,
  inheritance = .inheritanceIndex(phasedData)
)
```

Arguments

`received`	Named vector of the allele each child received from the heterozygous parent, named by child ID
`alleles`	Allele 1 and allele 2 of the SNP
`phasedData`	Data frame with phased informative SNPs
`hetId`	ID of the heterozygous parent
`hetSuffix`	Suffix of the children's haplotype columns from that parent ("Pat" or "Mat")
`snpPos`	Physical position of the SNP
`inheritance`	Inheritance index of `phasedData` (see `.inheritanceIndex()`)

Value

The distinct alleles implied for the parent's haplotype 1 by the children that give evidence

.inheritanceIndex *Index of Which Haplotype Each Child Inherited*

Description

Gives the same answer as `.inheritedHaplotype()`, but scans the phased SNPs once per child and parent instead of once per SNP looked up. For each child and parent it keeps the usable phased SNPs (the parent's two haplotypes differ and the child's allele from that parent is known), sorted by position, with the haplotype the child inherited at each; a lookup is then a binary search (`findInterval()`). Where several usable SNPs share the nearest position, the first in `phasedData` is used, as in `.inheritedHaplotype()`.

Usage

```
.inheritanceIndex(phasedData)
```

Arguments

phasedData	Data frame with phased informative SNPs

Value

A function of `childCol` (the child's haplotype column for the parent, e.g. `"F1C1Pat"`), `parentId` and `snpPos` that returns 1, 2 or NA, as `.inheritedHaplotype()` does. Each child and parent pair is indexed the first time it is looked up.

.inheritedHaplotype *Haplotype a Child Inherited Around a SNP*

Description

Which haplotype (1 or 2) of a heterozygous parent a child inherited around a SNP, read from the nearest phased SNP on each side where the parent's two haplotypes differ and the child's allele from that parent is known.

Usage

```
.inheritedHaplotype(childAlleles, hap1, hap2, positions, snpPos)
```

Arguments

childAlleles	Child's alleles from the parent at the phased SNPs
hap1, hap2	Parent's haplotypes at the phased SNPs
positions	Physical positions of the phased SNPs
snpPos	Physical position of the SNP

Value

1 or 2, or NA if the two sides disagree (the child has a crossover between them) or there is no usable SNP. Near a chromosome end, the one available side is used.

.isHomozygous *Classify Genotype as Homozygous or Heterozygous*

Description

Converts numeric genotype codes to binary (1 = homozygous, 0 = heterozygous)

Usage

```
.isHomozygous(genoVec)
```

Arguments

genoVec	Numeric vector of genotype codes (0=AA, 1=AB, 2=BB, NA=missing)

Value

Integer vector: 1 for homozygous (0 or 2), 0 for heterozygous (1) or missing (NA)

.matchAlleleCode *Match Allele Code*

Description

Which allele (1 or 2) a child inherited from the heterozygous parent, given the genotypes of both parents. The homozygous parent passes on allele 2 `homGt / 2` times, so the heterozygous parent passed on allele 2 `childGt - homGt / 2` times; the code is 1 more than this.

Usage

```
.matchAlleleCode(parent, fatherGt, motherGt, childGt)
```

Arguments

parent	"father" or "mother": the heterozygous parent
fatherGt	Father genotype(s) (0, 1, or 2)
motherGt	Mother genotype(s) (0, 1, or 2)
childGt	Child genotype(s) (0, 1, or 2): a vector or a matrix with one row per SNP, recycled against `fatherGt` and `motherGt`

Value

Integer allele code(s) (1 or 2), with the dimensions of `childGt`; NA where the parent is not heterozygous, the other parent is not homozygous, or the child's genotype is inconsistent

.parentAlleleTables *Alleles Passed on by Each Parent*

Description

Alleles Passed on by Each Parent

Usage

```
.parentAlleleTables(genoAll, bim)
```

Arguments

genoAll	Informative genotypes, ordered father, mother, children
bim	BIM data frame

Value

List with pat and mat: for each parent, the nucleotide each child received at the SNPs where that parent is heterozygous, plus the SNPs where it is homozygous (marked hom = 0)

.phaseBothHeterozygous *Phase a SNP Where Both Parents Are Heterozygous*

Description

A homozygous child received the same allele from both parents, so its alleles are known. Each parent's phase follows from which of its haplotypes the homozygous children inherited around the SNP (see .impliedHap1Alleles()), and is left as NA if no child gives evidence or the children disagree. A heterozygous child is phased when the allele it received from at least one phased parent is known (see .transmittedAllele()); the other parent passed on the other allele. It is left as NA if the alleles from the two parents are known and equal, which a heterozygous child cannot have.

Usage

```
.phaseBothHeterozygous(
  childGeno,
  alleles,
  ids,
  snpPos,
  phasedData,
  inheritance = .inheritanceIndex(phasedData)
)
```

Arguments

childGeno	Named vector of child genotypes
alleles	Allele 1 and allele 2 of the SNP
ids	List with father, mother and children IDs
snpPos	Physical position of the SNP
phasedData	Data frame with phased informative SNPs
inheritance	Inheritance index of phasedData (see .inheritanceIndex())

Value

Named character vector of phased alleles

.phaseBothHomozygous *Phase a SNP Where Both Parents Are Homozygous*

Description

Each parent passes on its only allele, so every child with the expected genotype carries the father's allele on its paternal haplotype and the mother's allele on its maternal haplotype. Children with any other genotype, which indicates a genotyping error, are left as NA.

Usage

```
.phaseBothHomozygous(parentGeno, childGeno, alleles, ids)
```

Arguments

parentGeno	Father and mother genotypes (0 or 2)
childGeno	Named vector of child genotypes
alleles	Allele 1 and allele 2 of the SNP
ids	List with father, mother and children IDs

Value

Named character vector of phased alleles

.phaseChildren *Phase Child Haplotypes*

Description

Phase Child Haplotypes

Usage

```
.phaseChildren(actgCombPat, actgCombMat)
```

Arguments

actgCombPat	Paternal allele data frame
actgCombMat	Maternal allele data frame

Value

Data frame with phased children

.phasedVcfHeader *VCF Header for Phased Genotypes*

Description

VCF Header for Phased Genotypes

Usage

```
.phasedVcfHeader(sampleIds)
```

Arguments

sampleIds	Character vector of sample IDs

Value

VCFHeader object

.phaseListToVcf *Convert Phased Families to a VCF Object*

Description

Convert Phased Families to a VCF Object

Usage

```
.phaseListToVcf(phaseList, bim, fam)
```

Arguments

phaseList	Named list of phased DataFrames, one per family
bim	BIM data frame
fam	FAM data frame

Value

VCF object with phased GT for all family members

.phaseNonInformativeSnp *Phase One Non-Informative SNP*

Description

Phase One Non-Informative SNP

Usage

```
.phaseNonInformativeSnp(
  snpId,
  genoSubset,
  bim,
  ids,
  fatherCol,
  motherCol,
  phasedData,
  bimIdx = match(snpId, bim$snp),
  rowIdx = match(snpId, rownames(genoSubset)),
  inheritance = .inheritanceIndex(phasedData)
)
```

Arguments

snpId	SNP identifier
genoSubset	Full genotype matrix for the family
bim	BIM data frame with SNP information
ids	List with father, mother and children IDs
fatherCol, motherCol	Column indices for the parents in genoSubset
phasedData	Data frame with phased informative SNPs
bimIdx	Row of bim for the SNP. Pass it when phasing many SNPs so that the lookup is done once for all of them, not once per SNP. Defaults to match(snpId, bim$snp).
rowIdx	Row of genoSubset for the SNP, passed for the same reason. Defaults to match(snpId, rownames(genoSubset)).
inheritance	Inheritance index of phasedData (see .inheritanceIndex()), passed for the same reason. Defaults to .inheritanceIndex(phasedData).

Value

Named character vector with one value per column of phasedData: the SNP ID, its location and the phased alleles, NA where unknown. All NA for a SNP that is not in the BIM file.

.phaseOneHeterozygous *Phase a SNP Where One Parent Is Heterozygous*

Description

The homozygous parent's haplotypes both carry its allele. The allele each child received from the heterozygous parent follows from the genotypes (see .matchAlleleCode()); children with a missing or inconsistent genotype are left as NA. The heterozygous parent's phase follows from which of its haplotypes each child inherited around the SNP (see .inheritedHaplotype()): that haplotype carries the allele the child received. The parent is left as NA if no child gives evidence or the children disagree.

Usage

```
.phaseOneHeterozygous(
  parentGeno,
  childGeno,
  alleles,
  ids,
  snpPos,
  phasedData,
  inheritance = .inheritanceIndex(phasedData)
)
```

Arguments

parentGeno	Father and mother genotypes, one of them 1
childGeno	Named vector of child genotypes
alleles	Allele 1 and allele 2 of the SNP
ids	List with father, mother and children IDs
snpPos	Physical position of the SNP
phasedData	Data frame with phased informative SNPs
inheritance	Inheritance index of phasedData (see .inheritanceIndex())

Value

Named character vector of phased alleles

.phaseParent *Phase Parent Haplotypes Using Crossover Information*

Description

Phase Parent Haplotypes Using Crossover Information

Usage

```
.phaseParent(actgMatrix, xoDetectMatrix, bim, parentId)
```

Arguments

actgMatrix	Allele matrix for parent
xoDetectMatrix	Crossover detection results
bim	BIM data frame
parentId	Parent identifier

Value

Data frame with two phased haplotypes

.phaseSingleFamily *Phase a Single Family*

Description

Phase a Single Family

Usage

```
.phaseSingleFamily(familyId, geno, fam, bim, xoPat, xoMat)
```

Arguments

familyId	Family identifier
geno	Numeric genotype matrix
fam	FAM data frame
bim	BIM data frame
xoPat	Paternal crossover data frame
xoMat	Maternal crossover data frame

Value

DataFrame with phased haplotypes, or NULL if the family cannot be phased

.rohResultDataFrame *ROH Results as a DataFrame*

Description

ROH Results as a DataFrame

Usage

```
.rohResultDataFrame(rohList, withCm)
```

Arguments

rohList	List of character matrices from .detectRohSample(), one per sample with ROH
withCm	Logical; whether the matrices include genetic positions

Value

DataFrame with one row per ROH (no rows if rohList is empty)

.selectSamples *Select Samples*

Description

Select Samples

Usage

```
.selectSamples(genoMatrix, sampleIds)
```

Arguments

genoMatrix	Genotype matrix with samples in columns
sampleIds	Character vector of sample IDs, or NULL for all samples

Value

The genotype matrix for the requested samples; stops if none of them are found

.transmittedAllele *Allele a Child Received from a Phased Parent*

Description

Allele a Child Received from a Phased Parent

Usage

```
.transmittedAllele(
  child,
  suffix,
  parentId,
  parentHaps,
  phasedData,
  snpPos,
  inheritance = .inheritanceIndex(phasedData)
)
```

Arguments

child	Child identifier
suffix	Suffix of the child's haplotype columns from this parent ("Pat" or "Mat")
parentId	Parent identifier
parentHaps	The parent's alleles on haplotypes 1 and 2 at the SNP, or NULL if the parent is not phased there
phasedData	Data frame with phased informative SNPs
snpPos	Physical position of the SNP
inheritance	Inheritance index of phasedData (see .inheritanceIndex())

Value

The allele on the haplotype the child inherited around the SNP (see `.inheritedHaplotype()`), or NA if unknown

---

.undoCrossovers *Undo a Child's Crossovers*

Description

The alleles a child received from a parent, switched to the parent's other allele wherever the child carries the parent's other haplotype, so that they all lie on one parental haplotype.

Usage

```
.undoCrossovers(actgMatrix, childName, xoDetectMatrix, bim)
```

Arguments

`actgMatrix`	Allele table for the parent (see .phaseParent())
`childName`	Child identifier (a column of actgMatrix)
`xoDetectMatrix`	Crossover data frame for the parent
`bim`	BIM data frame

Value

Character vector of alleles, one per row of actgMatrix

.vcfSampleData *Sample Information for the VCF*

Description

Sample Information for the VCF

Usage

```
.vcfSampleData(sampleIds, fam)
```

Arguments

`sampleIds`	Character vector of sample IDs
`fam`	FAM data frame

Value

DataFrame with familyId, fatherId, motherId, sex and affected from the FAM file, one row per sample

`crossoverData` *Example Crossover Data*

Description

Pre-computed maternal and paternal crossover events from the simCEU dataset. Used for testing and demonstrating haplotype phasing functionality.

Format

GRanges objects stored as RDS files with metadata columns:

childId Child identifier

familyId Family identifier

startSnp SNP name at crossover interval start

finishSnp SNP name at crossover interval end

startCm Genetic position at interval start (cM)

finishCm Genetic position at interval end (cM)

Details

Two crossover datasets are provided:

- `xoMat.rds` - Maternal crossover events
- `xoPat.rds` - Paternal crossover events

These objects contain detected crossover events from the families with three or more children in the simCEU dataset. Crossovers were identified using:

```
# Maternal crossovers
xoMat <- xoDetect(
  plinkFile = "simCEU",
  mapFile = "female_chr4.txt",
  familySize = 3,
  parent = "mother",
  snpFilter = 5,
  cmFilter = 1
)

# Paternal crossovers
xoPat <- xoDetect(
  plinkFile = "simCEU",
  mapFile = "male_chr4.txt",
  familySize = 3,
  parent = "father",
  snpFilter = 5,
  cmFilter = 1
)
```

Access the crossover data using:

```
dataPath <- system.file("extdata", package = "inferRecom")
xoMat <- readRDS(file.path(dataPath, "xoMat.rds"))
xoPat <- readRDS(file.path(dataPath, "xoPat.rds"))
```

Value

No R object is returned. Access files via `system.file("extdata", package = "inferRecom")`.

Source

Created by `inst/scripts/make-crossovers.R`.

`geneticMaps-data` *Genetic Maps for Chromosome 4*

Description

Female, male and sex-averaged genetic maps for chromosome 4 (`female_chr4.txt`, `male_chr4.txt`, `sexavg_chr4.txt`).

Format

Tab-delimited text file with columns:

chr Chromosome identifier (chr4)

pos Physical position in base pairs (hg19/GRCh37)

rate Recombination rate (cM/Mb) at this position

cM Cumulative genetic distance in centiMorgans from chromosome start

Access the files using:

```
femaleMap <- read.delim(file.path(dataPath, "female_chr4.txt"))
```

Value

No R object is returned. Access files via `system.file("extdata", package = "inferRecom")`.

Source

The chromosome 4 files of the refined European maps of Bherer, et al. (2017), unchanged; see `inst/scripts/make-genetic-maps.R`.

References

Bhérer, C., Campbell, C. L., & Auton, A. (2017). Refined genetic maps reveal sexual dimorphism in human meiotic recombination at multiple scales. Nature communications, 8(1), 14994.

hzRun *Detect Runs of Homozygosity in PLINK Genotype Data*

Description

This function identifies runs of homozygosity (ROH) in PLINK genotype data based on minimum physical length and SNP count thresholds. It can optionally filter for specific samples, apply a minimum genetic distance criterion, and separate results by case/control status.

Usage

```
hzRun(
  plinkFile,
  mapFile = NULL,
  minMb = 1,
  minSnps = 5,
  minCm = NULL,
  rsOnly = FALSE,
  sampleIds = NULL,
  caseControl = FALSE,
  BPPARAM = SerialParam()
)
```

Arguments

`plinkFile`	Character string; path to the PLINK prefix (without extensions).
`mapFile`	Character string or NULL; path to a tab-delimited genetic map file containing columns: `chr`, `pos`, `cM`. If NULL, cM filtering is not available. Default is NULL.
`minMb`	Numeric; minimum physical length of ROH in megabases. Default is 1.
`minSnps`	Integer; minimum number of consecutive homozygous SNPs required for an ROH. Default is 5.
`minCm`	Numeric or NULL; minimum genetic length of ROH in centiMorgans. If NULL, no genetic length filtering is applied. Requires mapFile to be specified. Default is NULL.
`rsOnly`	Logical; whether to restrict analysis to SNPs with names beginning with "rs". Default is FALSE.
`sampleIds`	Character vector or NULL; specific sample IDs to analyze. If NULL, all samples are analyzed. Default is NULL.
`caseControl`	Logical; if TRUE, function returns a GRangesList of case and control ROH, using the affected status in the FAM file with PLINK coding: 2 = affected (case); 1 = unaffected or missing (control). Default is FALSE.
`BPPARAM`	A BiocParallelParam object specifying parallel execution. Default is `SerialParam()` for serial execution. Use `MulticoreParam(workers = n)` for parallel processing across n cores.

Details

Runs of homozygosity (ROH) are continuous genomic segments where an individual is homozygous at all marker positions. ROH can indicate autozygosity (inheritance of identical haplotypes from a common ancestor) and are used to estimate inbreeding, identify disease-associated loci, and study population history.

This function identifies ROH by:

1. Converting genotypes to binary (1 = homozygous, 0 = heterozygous/missing)
2. Identifying runs using run-length encoding
3. Filtering runs by minimum SNP count and physical length
4. Optionally filtering by minimum genetic length (cM)

Genotypes from **snpStats** are coded as:

- 0 = homozygous reference (AA) - counted as homozygous
- 1 = heterozygous (AB) - breaks ROH
- 2 = homozygous alternate (BB) - counted as homozygous
- NA = missing - breaks ROH

Filtering strategy:

- Physical length (minMb): Always applied
- SNP count (minSnps): Always applied
- Genetic length (minCm): Applied only if mapFile is provided and minCm is not NULL

Parallel processing is performed by individual, improving efficiency for datasets with many samples.

Value

A GRanges object containing detected ROH with metadata columns: `sampleId`, `startSnp`, `finishSnp`, `numSnps`. If genetic map is provided, also includes `startCm`, `finishCm`. If `caseControl = TRUE`, returns a GRangesList with two elements: `case` and `control`. Returns an empty GRanges if no ROH are detected. Chromosome names use the UCSC style (e.g. `chr4`).

Examples

```
# Serial execution (default)

# Get path to example data
dataPath <- system.file("extdata", package = "inferRecom")
plinkFile <- file.path(dataPath, "simCEU")

# Detect ROH
rohData <- hzRun(
    plinkFile = plinkFile,
    minMb = 1,
    minSnps = 50
)

# View results
head(rohData)
```

```
# Analyze specific individuals
sampleIds <- c("F3C1", "F4C3", "F5P2")
rohSubset <- hzRun(
    plinkFile = plinkFile,
    minMb = .5,
    minSnps = 50,
    sampleIds = sampleIds
)

rohSubset

# Separate by case/control status
rohCc <- hzRun(
    plinkFile = plinkFile,
    minMb = .5,
    minSnps = 50,
    caseControl = TRUE
)

rohCases <- rohCc$case
rohControls <- rohCc$control

rohCases
rohControls
```

`simCEU-data` *Example PLINK Dataset - simCEU*

Description

Simulated pedigrees based on HapMap CEU (Utah residents with Northern and Western European ancestry) genotypes, for demonstrating crossover detection, phasing and ROH analysis. These data are for package development and examples only and do not reflect any biological principle: recombination was simulated with a uniform map of 100 cM across the region, far above real recombination rates, so that the short region contains crossovers.

Format

PLINK binary format files (.bed, .bim, .fam) containing:

Families 8 nuclear families, 40 individuals: three families with 2 children, three with 3, one with 4 and one with 5

SNPs 1162 SNPs on chromosome 4 between 78.0 and 79.3 Mb (GRCh37)

Genotypes Founder haplotypes sampled from HapMap CEU genotypes, with recombination simulated in each meiosis

Details

The dataset includes three PLINK binary files:

- `simCEU.bed` - Binary genotype data
- `simCEU.bim` - Variant information (SNP IDs, positions, alleles)

- `simCEU.fam` - Sample information (family IDs, relationships, sex and affected status). One child per family is affected (PLINK phenotype 2); everyone else is unaffected (1).

Access the files using:

```
dataPath <- system.file("extdata", package = "inferRecom")
plinkFile <- file.path(dataPath, "simCEU")
```

Value

No R object is returned. Access files via `system.file("extdata", package = "inferRecom")`.

Source

Simulated with the R package `sim1000G` from HapMap CEU genotypes (2010 release); see `inst/scripts/make-simCEU` and `inst/scripts/README.md`.

References

The International HapMap 3 Consortium (2010). Integrating common and rare genetic variation in diverse human populations. Nature, 467, 52-58.

Dimitromanolakis, A., Xu, J., Krol, A., & Briollais, L. (2019). sim1000G: a user-friendly genetic variant simulator in R for unrelated individuals and family-based designs. BMC bioinformatics, 20(1), 26.

`xoDetect` *Detect Meiotic Recombinations from PLINK Genotype Data*

Description

This function identifies recombination events in family-based genotype data, using PLINK-formatted input files and a genetic map. It supports 2- or 3-child family structures, and can optionally write results to disk if an output path is provided.

Usage

```
xoDetect(
  plinkFile,
  mapFile,
  familySize = c(2, 3),
  parent = c("mother", "father"),
  rsOnly = FALSE,
  snpFilter = 5,
  cmFilter = 1,
  caseControl = FALSE,
  out = NULL,
  BPPARAM = SerialParam()
)
```

Arguments

plinkFile	Character string; path to the PLINK prefix (without extensions).
mapFile	Character string; path to a tab-delimited map file containing columns: chr, pos, rate, cM.
familySize	Integer; either 2 or 3, specifying the family pedigree size.
parent	Character; either "mother" or "father", specifying the parent to trace crossovers from.
rsOnly	Logical; whether to restrict to SNPs with names beginning with "rs". Default is FALSE.
snpFilter	Integer; minimum number of informative SNPs separating two putative crossovers in the same child. A crossover that is reversed within fewer SNPs, typically because of genotyping error, is removed together with its reversal. Default is 5.
cmFilter	Numeric; minimum genetic distance (cM) separating putative crossovers. Default is 1.
caseControl	Logical; if TRUE, function returns a GRangesList of case and control crossovers for 3+ child families, using the affected status of each child in the FAM file with PLINK coding: 2 = affected (case); 1 = unaffected or missing (control). Default is FALSE.
out	Character or NULL; if provided, output is written as a CSV at this path.
BPPARAM	A BiocParallelParam object specifying parallel execution. Default is SerialParam() for serial execution. Use MulticoreParam(workers = n) for parallel processing across n cores.

Details

This function reads PLINK genotype data (via **snpStats**) and infers loci where meiotic recombination has occurred, resolved to the nearest informative SNPs. The algorithm identifies informative marker configurations where parents are heterozygous/homozygous and filters for Mendelian consistency before detecting inheritance state changes that indicate crossovers.

For 3-child families, children are analyzed in triples to allow identification of the specific recombinant child. For 2-child families, state changes between the two siblings are used to infer crossover locations, but the individual cannot be determined.

If out is provided, the resulting table is written to disk as UTF-8 encoded CSV without row names.

Parallel processing is performed by family, improving efficiency for datasets with many families.

Value

A GRanges listing detected crossover intervals with metadata columns: childId (in 3-child families), familyId, startSnp, finishSnp, startPos, finishPos, startCm, finishCm. Chromosome names use the UCSC style (e.g. chr4).

Examples

```
# Serial execution (default)

# Load example data
dataPath <- system.file("extdata", package = "inferRecom")
plinkFile <- file.path(dataPath, "simCEU")
```

```
# Basic maternal crossover detection
mapFemale <- file.path(dataPath, "female_chr4.txt")
xoMat3 <- xoDetect(
    plinkFile = plinkFile,
    mapFile = mapFemale,
    familySize = 3,
    parent = "mother"
)

# View results
xoMat3

# Basic 2-child family
xoMat2 <- xoDetect(
    plinkFile = plinkFile,
    mapFile = mapFemale,
    familySize = 2,
    parent = "mother"
)

# View results
xoMat2

# Maternal crossover detection with case/control separation
mapFemale <- file.path(dataPath, "female_chr4.txt")
xoMat3CC <- xoDetect(
    plinkFile = plinkFile,
    mapFile = mapFemale,
    familySize = 3,
    parent = "mother",
    caseControl = TRUE
)

# View results
xoMat3CC$case
xoMat3CC$control
```

---

xoPhase *Phase Haplotypes Using Crossover Information*

Description

This function phases parental and child haplotypes based on detected crossover events from maternal and paternal meioses.

Usage

```
xoPhase(
  plinkFile,
  xoDetectPaternal,
  xoDetectMaternal,
  famIds = NULL,
  rsOnly = TRUE,
```

```
  BPPARAM = SerialParam()
)
```

Arguments

`plinkFile`	Character string; path to the PLINK prefix (without extensions).
`xoDetectPaternal`	GRanges; output from `xoDetect()` for paternal crossovers (parent = "father").
`xoDetectMaternal`	GRanges; output from `xoDetect()` for maternal crossovers (parent = "mother").
`famIds`	Character vector or NULL; specific family IDs to phase. If NULL, all families with 5+ members are analyzed. Default is NULL.
`rsOnly`	Logical; whether to restrict to SNPs with names beginning with "rs". Default is TRUE.
`BPPARAM`	A BiocParallelParam object specifying parallel execution. Default is `SerialParam()` for serial execution. Use `MulticoreParam(workers = n)` for parallel processing across n cores.

Details

This function performs haplotype phasing using family-based genetic data and detected crossover events. The phasing process:

1. Identifies informative SNPs where parents are heterozygous/homozygous
2. Assigns alleles (1 or 2) to children based on inheritance
3. Converts allele codes to nucleotides (A, C, G, T)
4. Imputes homozygous sites from the other parent
5. Phases parental haplotypes using crossover breakpoints
6. Adds back non-informative SNPs with inferred genotypes where possible
7. Outputs diploid genotypes for all family members across all SNPs

Non-informative SNPs are handled as follows. In every case, a child whose genotype is missing or inconsistent with its parents' genotypes is left unphased.

- Both parents homozygous: each parent passes on its only allele.
- Both parents heterozygous: homozygous children carry their only allele. Each parent is phased from which of its haplotypes the homozygous children inherited at the nearest phased SNPs on both sides, and left unknown if that evidence is missing or conflicting. A heterozygous child is phased from the haplotypes it inherited from the phased parents.
- One parent heterozygous, one homozygous (informative SNPs left out because of a missing or inconsistent genotype): each child's alleles follow from the genotypes. The heterozygous parent is phased from which of its haplotypes the children inherited at the nearest phased SNPs on both sides, and left unknown if that evidence is missing or conflicting.

The function requires crossover detection results from both parents as GRanges objects. Run `xoDetect()` separately for maternal and paternal crossovers before using this function.

Parallel processing is performed by family, improving efficiency for datasets with many families.

Value

A VCF object with one row per SNP and one column per family member, for all phased families. The GT genotype field holds phased genotypes (e.g. "0|1"), coded against REF (PLINK allele 2, BIM column 6) and ALT (PLINK allele 1, BIM column 5), the convention of plink2 and of PLINK 1.9 with --keep-allele-order. A BIM file made by PLINK 1.9 without that flag has the minor allele first, so REF is then the major allele, not necessarily the reference allele; "." marks an allele that could not be phased. For children the first allele is paternal and the second maternal; for parents the alleles are the two phased haplotypes. colData holds familyId, fatherId, motherId, sex and affected from the FAM file. Families without phase information are omitted with a message. Chromosome names use the UCSC style (e.g. chr4). Use VariantAnnotation::writeVcf() to write the result to a file.

Examples

```
# Load example data
dataPath <- system.file("extdata", package = "inferRecom")
plinkFile <- file.path(dataPath, "simCEU")
mapFemale <- file.path(dataPath, "female_chr4.txt")
mapMale <- file.path(dataPath, "male_chr4.txt")

# Detect crossovers for both parents (returns GRanges objects)
xoPat <- xoDetect(
    plinkFile = plinkFile,
    mapFile = mapMale,
    familySize = 3,
    parent = "father"
)
xoMat <- xoDetect(
    plinkFile = plinkFile,
    mapFile = mapFemale,
    familySize = 3,
    parent = "mother"
)

# Phase haplotypes (serial execution, default)
phased <- xoPhase(
    plinkFile = plinkFile,
    xoDetectPaternal = xoPat,
    xoDetectMaternal = xoMat
)

phased

# Phased genotypes for the members of family F6
f6 <- colData(phased)$familyId == "F6"
head(geno(phased)$GT[, f6])

# Phase haplotypes for subset of families
phasedSubset <- xoPhase(
    plinkFile = plinkFile,
    xoDetectPaternal = xoPat,
    xoDetectMaternal = xoMat,
    famIds = c("F4", "F6")
)

phasedSubset
```

```
# Write to a VCF file
vcfFile <- tempfile(fileext = ".vcf")
writeVcf(phasedSubset, vcfFile)
```

Index

∗ **internal**
  .addHomozygousSNPs, 3
  .addNonInformativeSNPs, 3
  .alignSnpData, 4
  .alignToSnps, 4
  .assignAlleleCodes, 5
  .bimFromMap, 5
  .checkCrossovers, 6
  .checkSameChromosome, 6
  .completeHeterozygote, 7
  .convertToNucleotides, 7
  .detectRohAllSamples, 8
  .detectRohSample, 8
  .familiesToPhase, 9
  .familyGenotypes, 10
  .getInformativeGenotypes, 10
  .getOtherAllele, 11
  .getSnpLocations, 11
  .impliedHap1Alleles, 12
  .inheritanceIndex, 12
  .inheritedHaplotype, 13
  .isHomozygous, 14
  .matchAlleleCode, 14
  .parentAlleleTables, 15
  .phaseBothHeterozygous, 15
  .phaseBothHomozygous, 16
  .phaseChildren, 17
  .phaseListToVcf, 18
  .phaseNonInformativeSnp, 18
  .phaseOneHeterozygous, 19
  .phaseParent, 20
  .phaseSingleFamily, 21
  .phasedVcfHeader, 17
  .rohResultDataFrame, 21
  .selectSamples, 22
  .transmittedAllele, 22
  .undoCrossovers, 23
  .vcfSampleData, 23
.addHomozygousSNPs, 3
.addNonInformativeSNPs, 3
.alignSnpData, 4
.alignToSnps, 4
.assignAlleleCodes, 5
.bimFromMap, 5
.checkCrossovers, 6
.checkSameChromosome, 6
.completeHeterozygote, 7
.convertToNucleotides, 7
.detectRohAllSamples, 8
.detectRohSample, 8
.familiesToPhase, 9
.familyGenotypes, 10
.getInformativeGenotypes, 10
.getOtherAllele, 11
.getSnpLocations, 11
.impliedHap1Alleles, 12
.inheritanceIndex, 12
.inheritedHaplotype, 13
.isHomozygous, 14
.matchAlleleCode, 14
.parentAlleleTables, 15
.phaseBothHeterozygous, 15
.phaseBothHomozygous, 16
.phaseChildren, 17
.phaseListToVcf, 18
.phaseNonInformativeSnp, 18
.phaseOneHeterozygous, 19
.phaseParent, 20
.phaseSingleFamily, 21
.phasedVcfHeader, 17
.rohResultDataFrame, 21
.selectSamples, 22
.transmittedAllele, 22
.undoCrossovers, 23
.vcfSampleData, 23

crossoverData, 24

geneticMaps-data, 25
GRanges, *24*, *27*, *30*, *32*
GRangesList, *26*, *27*, *30*

hzRun, 26

simCEU-data, 28

VCF, *33*

xoDetect, 29
xoPhase, 31