Package 'ibdsim2'

August 4, 2026

Type Package

Title Simulation of Chromosomal Regions Shared by Family Members

Version 2.3.3

Description Simulation of chromosomal segments shared identical-by-descent (IBD) among pedigree members. Using sex-specific recombination rates along the human genome (Halldorsson et al. (2019) <doi:10.1126/science.aau1043>), phased chromosomes are simulated for all pedigree members. Applications include calculation of realised relatedness coefficients and IBD segment distributions. 'ibdsim2' is part of the 'pedsuite' collection of packages for pedigree analysis. A detailed presentation of the 'pedsuite', including a separate chapter on 'ibdsim2', is available in the book 'Pedigree analysis in R' (Vigeland, 2021, ISBN:9780128244302). A 'Shiny' app for visualising and comparing IBD distributions is available at <https://magnusdv.shinyapps.io/ibdsim2-shiny/>.

License GPL-3

URL https://github.com/magnusdv/ibdsim2, https://magnusdv.github.io/pedsuite/, https://magnusdv.shinyapps.io/ibdsim2-shiny/

Depends R (>= 4.2.0), pedtools (>= 2.8.0)

Imports Rcpp, ggplot2, glue, ribd (>= 1.7.1)

Suggests MASS, patchwork, shiny (>= 1.9.0), shinyWidgets, shinyjs, testthat, zip

LinkingTo Rcpp

Config/roxygen2/version 8.0.0

Encoding UTF-8

Language en-GB

LazyData true

NeedsCompilation yes

Author Magnus Dehli Vigeland [aut, cre] (ORCID: <https://orcid.org/0000-0002-9134-4962>)

Maintainer Magnus Dehli Vigeland <m.d.vigeland@medisin.uio.no>

Repository CRAN

Date/Publication 2026-08-04 20:00:10 UTC

Contents

convertPos . . . 2
customMap . . . 3
estimateCoeffs . . . 4
extractIds . . . 8
findPattern . . . 8
haploDraw . . . 10
ibdsim . . . 12
karyoDiploid . . . 14
karyogram2 . . . 15
karyoHaploid . . . 16
launchApp . . . 18
legacy_decode19 . . . 18
loadMap . . . 19
maplengths . . . 21
plotSegmentDistribution . . . 22
profileSimIBD . . . 25
realised . . . 26
segmentStats . . . 28
uniformMap . . . 29
zeroIBD . . . 30

Index **32**

`convertPos` *Conversion of genetic map positions*

Description

Convert between physical position (in megabases) and genetic position (centiMorgan) given a chromosome map. Linear interpolation is used between map points. Values outside the map are returned as 0, the terminal genetic position, or NA, depending on the conversion direction.

Usage

```
convertPos(
  chrom = NULL,
  Mb = NULL,
  cM = NULL,
  map = "decode19",
  sex = c("average", "male", "female")
)
```

Arguments

chrom	(Optional) A vector of chromosome labels.
Mb	A vector of physical positions (in Mb), or NULL.
cM	A vector of genetic positions (in cM), or NULL.
map	A genomeMap, a chromMap, or a data frame with columns Mb and cM. By default, loadMap("decode19") is used.
sex	Either "average", "male" or "female".

Value

A vector of the same length as the input.

Examples

```
# Chromosome 1 of the built-in recombination map
map = loadMap(chrom = 1)[[1]]
head(map$male)

# Conversion Mb -> cM
phys = 1:5
gen = convertPos(Mb = phys, map = map, sex = "male")
gen

# Convert back (note that the first position cannot be recovered in this example)
convertPos(cM = gen, map = map, sex = "male")
```

---

customMap *Custom recombination map*

Description

Create custom recombination maps for use in ibdsim().

Usage

```
customMap(x)
```

Arguments

x	A data frame or matrix. See details for format specifications.

Details

The column names of x must include either

- chrom, mb and cm (sex-averaged map)

or

- chrom, mb, male and female (sex-specific map)

Upper-case letters are allowed in these names. The mb column should contain physical positions in megabases, while cm, male, female give the corresponding genetic position in centiMorgans.

Value

An object of class genomeMap.

See Also

uniformMap(), loadMap()

Examples

```
# A map including two chromosomes.
df1 = data.frame(chrom = c(1, 1, 2, 2),
                 mb = c(0, 2, 0, 5),
                 cm = c(0, 3, 0, 6))
map1 = customMap(df1)
map1

# Use columns "male" and "female" to make sex-specific maps
df2 = data.frame(chrom = c(1, 1, 2, 2),
                 mb = c(0, 2, 0, 5),
                 male = c(0, 3, 0, 6),
                 female = c(0, 4, 0, 7))
map2 = customMap(df2)
map2
```

---

estimateCoeffs *Estimation of one- and two-locus relatedness coefficients*

Description

Estimate by simulation various relatedness coefficients, and two-locus versions of the same coefficients, for a given recombination rate. The current implementation covers inbreeding coefficients, kinship coefficients, IBD (kappa) coefficients between noninbred individuals, and condensed identity coefficients. These functions are primarily meant as tools for validating exact algorithms, e.g., as implemented in the ribd package.

Usage

```
estimateInbreeding(x, id, Nsim, Xchrom = FALSE, verbose = FALSE, ...)

estimateTwoLocusInbreeding(
  x,
  id,
  rho = NULL,
  cM = NULL,
  Nsim,
  Xchrom = FALSE,
  verbose = FALSE,
  ...
)

estimateKinship(x, ids, Nsim, Xchrom = FALSE, verbose = FALSE, ...)

estimateTwoLocusKinship(
  x,
  ids,
  rho = NULL,
  cM = NULL,
  Nsim,
  Xchrom = FALSE,
  verbose = FALSE,
  ...
)

estimateKappa(x, ids, Nsim, Xchrom = FALSE, verbose = FALSE, ...)

estimateTwoLocusKappa(
  x,
  ids,
  rho = NULL,
  cM = NULL,
  Nsim,
  Xchrom = FALSE,
  verbose = FALSE,
  ...
)

estimateIdentity(x, ids, Nsim, Xchrom = FALSE, verbose = FALSE, ...)

estimateTwoLocusIdentity(
  x,
  ids,
  rho = NULL,
  cM = NULL,
  Nsim,
```

```
  Xchrom = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

x	A pedigree in the form of a pedtools::ped() object.
id, ids	A vector of one or two ID labels.
Nsim	The number of simulations.
Xchrom	A logical indicating if the loci are X-linked or autosomal.
verbose	A logical.
...	Further arguments passed on to ibdsim(), e.g. seed.
rho	A scalar in the interval [0, 0.5]: the recombination fraction between the two loci, converted to centiMorgans using Haldane's map function: cM = -50 * log(1 - 2 * rho). Either rho or cM (but not both) must be non-NULL.
cM	A non-negative number: the genetic distance between the two loci, given in centiMorgans. Either rho or cM (but not both) must be non-NULL.

Details

In the following, let L1 and L2 denote two arbitrary autosomal loci with recombination rate ρ, and let A and B be members of the pedigree x.

The *two-locus inbreeding coefficient* $f_2(\rho)$ of A is defined as the probability that A is autozygous at both L1 and L2 simultaneously.

The *two-locus kinship coefficient* $\phi_2(\rho)$ of A and B is defined as the probability that a random gamete emitted from A, and a random gamete emitted from B, contain IBD alleles at both L1 and L2.

The *two-locus kappa coefficient* $\kappa_{ij}(\rho)$, for $i, j = 0, 1, 2$, of noninbred A and B, is the probability that A and B share exactly i alleles IBD at L1, and exactly j alleles IBD at L2.

The *two-locus identity coefficient* Δ_{ij}, $i, j = 1, ..., 9$ is defined for any (possibly inbred) A and B, as the probability that A and B are in identity state i at L1, and state j at L2. This uses the conventional ordering of the nine condensed identity states. For details, see for instance the GitHub page of the ribd package.

Value

estimateInbreeding(): a single probability.

estimateKinship(): a single probability.

estimateTwoLocusInbreeding(): a single probability.

estimateKappa(): a numeric vector of length 3, with the estimated κ coefficients.

estimateTwoLocusKappa(): a numeric 3*3 matrix, with the estimated values of κ_{ij}, for $i, j = 0, 1, 2$.

estimateIdentity(): a numeric vector of length 9, with the estimated identity coefficients.

estimateTwoLocusIdentity(): a numeric 9*9 matrix, with the estimated values of Δ_{ij}, for $i, j = 1, ..., 9$.

Examples

```
############################
### Two-locus inbreeding ###
############################

x = cousinPed(0, child = TRUE)
rho = 0.25
Nsim = 10 # Increase!
estimateTwoLocusInbreeding(x, id = 5, rho = rho, Nsim = Nsim, seed = 123)

# Exact:
ribd::twoLocusInbreeding(x, id = 5, rho = rho)

########################################
### Two-locus kappa:                 ###
### Grandparent vs half sib vs uncle ###
########################################

# These are indistinguishable with unlinked loci, see e.g.
# pages 182-183 in Egeland, Kling and Mostad (2016).
# In the following, each simulation approximation is followed
# by its exact counterpart.

rho = 0.25; R = .5 * (rho^2 + (1-rho)^2)
Nsim = 10 # Should be increased to at least 10000

# Grandparent/grandchild
G = linearPed(2); G.ids = c(1,5); # plot(G, hatched = G.ids)
estimateTwoLocusKappa(G, G.ids, rho = rho, Nsim = Nsim, seed = 123)[2,2]
.5*(1-rho) # exact

# Half sibs
H = halfSibPed(); H.ids = c(4,5); # plot(H, hatched = H.ids)
estimateTwoLocusKappa(H, H.ids, rho = rho, Nsim = Nsim, seed = 123)[2,2]
R # exact

# Uncle
U = avuncularPed(); U.ids = c(3,6); # plot(U, hatched = U.ids)
estimateTwoLocusKappa(U, U.ids, rho = rho, Nsim = Nsim, seed = 123)[2,2]
(1-rho) * R + rho/4 # exact

# Exact calculations by ribd:
# ribd::twoLocusIBD(G, G.ids, rho = rho, coefs = "k11")
# ribd::twoLocusIBD(H, H.ids, rho = rho, coefs = "k11")
# ribd::twoLocusIBD(U, U.ids, rho = rho, coefs = "k11")

#########################
### Two-locus Jacquard ###
#########################

x = fullSibMating(1)
rho = 0.25
```

```
Nsim = 10 # (increase to at least 10000)

estimateTwoLocusIdentity(x, ids = 5:6, rho = rho, Nsim = Nsim, seed = 123)

# Exact by ribd:
# ribd::twoLocusIdentity(x, ids = 5:6, rho = rho)
```

`extractIds` *Extract ID labels from simulation output*

Description

Extract ID labels from simulation output

Usage

```
extractIds(sim)
```

Arguments

`sim`	Output from `ibdsim()`

Value

A character vector

Examples

```
s = ibdsim(nuclearPed(2), N=1, ids = 3:4)
stopifnot(all(extractIds(s) == c("3", "4")))
```

`findPattern` *Find specific IBD patterns*

Description

Find segments satisfying a particular pattern of IBD sharing, in a list of IBD simulations.

Usage

```
findPattern(sims, pattern, merge = TRUE, cutoff = 0, unit = "mb")
```

Arguments

sims	A genomeSim object, or a list of such. Typically made by ibdsim().
pattern	A named list of vectors containing ID labels. Allowed names are autozygous, heterozygous, carriers, noncarriers.
merge	A logical, indicating if adjacent segments should be merged. Default: TRUE.
cutoff	A non-negative number. Segments shorter than this are excluded from the output. Default: 0.
unit	The unit of cutoff: either "mb" or "cm".

Details

For each simulation, this function extracts the subset of rows satisfying the allele sharing specified by pattern. That is, segments where, for some allele,

- all of pattern$autozygous are autozygous
- all of pattern$heterozygous have exactly one copy
- all of pattern$carriers have at least one copy
- none of pattern$noncarriers carry the allele.

Value

A matrix (if sims is a single genomeSim object), or a list of matrices.

See Also

segmentStats()

Examples

```
x = nuclearPed(3)
s = ibdsim(x, N = 1, map = uniformMap(M = 1), seed = 1729)

# Segments where some allele is shared by 3 and 4, but not 5
pattern = list(carriers = 3:4, noncarriers = 5)
findPattern(s, pattern)

# Exclude segments less than 7 cM
findPattern(s, pattern, cutoff = 7, unit = "cm")

# Visual confirmation:
haploDraw(x, s)
```

haploDraw *Draw haplotypes onto a pedigree plot*

Description

Visualise the IBD pattern of a single chromosome, by drawing haplotypes onto the pedigree.

Usage

```
haploDraw(
  x,
  ibd,
  chrom = NULL,
  ids = NULL,
  unit = "mb",
  L = NULL,
  pos = 1,
  cols = NULL,
  height = 4,
  width = 0.75,
  sep = 0.75,
  dist = 1,
  keep.par = FALSE,
  ...
)
```

Arguments

x	A ped object.
ibd	A genomeSim object, typically made by ibdsim().
chrom	A chromosome number, needed if ibd contains data from multiple chromosomes.
ids	A vector indicating for which pedigree members haplotypes should be drawn. If NULL (default), all individuals in ibd are included.
unit	Either "mb" (default) or "cm".
L	A positive number: the chromosome length. By default derived from ibd.
pos	A vector recycled to the number of individuals, indicating where haplotypes should be drawn relative to the pedigree symbols: 0 = no haplotypes; 1 = below; 2 = left; 3 = above; 4 = right. By default, all are placed below.
cols	A colour vector corresponding to the alleles in ibd.
height	The height of the haplotype rectangles in units of the pedigree symbol height. Default: 4.
width	The width of the haplotype rectangles in units of the pedigree symbol width. Default: 0.75.

sep	The separation between haplotypes within a pair, measured in pedigree symbol widths.
dist	The distance between pedigree symbols and the closest haplotype, measured in pedigree symbol widths.
keep.par	A logical, by default FALSE; passed on to plot.ped().
...	Further arguments passed on to plot.ped(), e.g. margins and cex. See ?plotmethods for a complete list.

Value

The pedigree plot object, invisibly.

Examples

```
##############################
# Example 1: A family quartet #
##############################

x = nuclearPed(2)
map = uniformMap(M = 1)
s = ibdsim(x, map = map, seed = 4276)

haploDraw(x, s)

# Custom colours and placements
haploDraw(x, s, cols = c(3,7,2,4), pos = c(2,4,2,4))

# Standard plot options apply
haploDraw(x, s, margins = 3, cex = 1.5, title = "Full sibs")


##########################
# Example 2: Autozygosity #
##########################

x = halfCousinPed(0, child = TRUE)
map = uniformMap(M = 1)
s = ibdsim(x, map = map, skipRecomb = c(1,3), seed = 2)

# Only include relevant individuals (skip 1 and 3)
haploDraw(x, s, ids = c(2,4,5,6), pos = c(1,2,4,4))

##############################
# Example 3: X-chromosomal sims
##############################

x = nuclearPed(2, sex = 2:1)
s = ibdsim(x, N = 1, map = uniformMap(M = 1, chrom = "X"), seed = 123)

haploDraw(x, s)
```

`ibdsim` *IBD simulation*

Description

This is the main function of the package, simulating the recombination process in each meiosis of a pedigree. The output summarises the IBD segments between all or a subset of individuals.

Usage

```
ibdsim(
  x,
  N = 1,
  ids = NULL,
  map = "decode19",
  model = c("chi", "haldane"),
  skipRecomb = NULL,
  simplify1 = TRUE,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`x`	A `pedtools::ped()` object.
`N`	A positive integer indicating the number of simulations.
`ids`	A vector of names indicating which pedigree members should be included in the output. Alternatively, a function taking `x` as input and returning such a vector, like `pedtools::leaves()`. By default (`ids = NULL`) everyone is included.
`map`	The genetic map to be used in the simulations: Allowed values are: • a `genomeMap` object, typically produced by `loadMap()` • a single `chromMap` object, for instance as produced by `uniformMap()` • a character, which is passed on to `loadMap()` with default parameters. Currently the only valid option is "decode19" (or abbreviations of this). Default: "decode19".
`model`	Either "chi" or "haldane", indicating the statistical model for recombination (see Details). Default: "chi".
`skipRecomb`	A vector of ID labels indicating individuals whose meioses should be simulated without recombination. (Each child will then receive a random strand of each chromosome.) Alternatively, a function taking `x` as input, e.g., `pedtools::founders()`. By default, recombination is skipped in founders who are uninformative for IBD sharing in the `ids` individuals.

simplify1	A logical, by default TRUE, removing the outer list layer when N = 1. See Value.
seed	An integer to be passed on to set.seed().
verbose	A logical.

Details

Each simulation starts by distributing unique alleles (labelled 1, 2, ...) to the pedigree founders, followed by a separate recombination process for each chromosome, according to the value of model:

- model = "chi" (default): This uses a renewal process along the four-strand bundle, with waiting times following a chi-square distribution.
- model = "haldane": In this model, crossover events are modelled as a Poisson process along each chromosome. (Faster, but less realistic.)

Recombination rates along each chromosome are determined by the map parameter. The default value ("decode19") loads a thinned version of the recombination map of the human genome published by Halldorsson et al. (2019). Note that autosomal and X-chromosomal maps cannot be combined in the same simulation.

In many applications, the fine-scale default map is not necessary, and may be replaced by simpler maps with constant recombination rates. See uniformMap() and loadMap() for ways to produce such maps.

The simulations support completely inbred founders; see pedtools::setFounderInbreeding().

Value

If N > 1, a list of N objects of class genomeSim.

If N = 1, the outer list layer is removed by default, which is typically desired in interactive use and in pipe chains. To enforce a list output in this case, add simplify1 = FALSE.

A genomeSim object is essentially a numerical matrix describing the allele flow through the pedigree in a single genome simulation. Each row corresponds to a chromosomal segment. The first 3 columns (chrom, startMB, endMB) describe the physical megabase location of the segment. Next come the centimorgan coordinates (startCM, endCM), which are computed from map by averaging the male and female values. Then follow the allele columns, two for each individual in ids, suffixed by ":p" and ":m" signifying the paternal and maternal alleles, respectively.

If ids has length 1, a column named Aut is added, whose entries are 1 for autozygous segments and 0 otherwise.

If ids has length 2, two columns are added:

- IBD : The IBD status of each segment, i.e., the number of alleles shared identical by descent. This is always either 0, 1, 2 or NA, where the latter appear if either individual (or both) of individuals is autozygous in the segment.
- Sigma : The condensed identity ("Jacquard") state of each segment, given as an integer in the range 1-9. The numbers correspond to the standard ordering of the condensed states. In particular, for non-inbred individuals, the states 9, 8, 7 correspond to IBD status 0, 1, 2 respectively.

References

Halldorsson et al. *Characterizing mutagenic effects of recombination through a sequence-level genetic map.* Science 363, no. 6425 (2019).

Examples

```
### Example 1: Half siblings ###

hs = halfSibPed()
sim = ibdsim(hs, map = uniformMap(M = 1), seed = 10)
sim

# Plot haplotypes
haploDraw(hs, sim)

### Example 2: Full sib mating ###

x = fullSibMating(1)
sim = ibdsim(x, ids = 5:6, map = uniformMap(M = 10), seed = 1)
head(sim)

# All 9 identity states are present
stopifnot(setequal(sim[, 'Sigma'], 1:9))
```

`karyoDiploid` *Diploid karyogram*

Description

Show chromosomal segments in a diploid karyogram

Usage

```
karyoDiploid(
  paternal,
  maternal,
  chrom = 1:22,
  col = c(paternal = "lightblue", maternal = "orange"),
  alpha = 1,
  bgcol = "gray95",
  title = NULL
)
```

Arguments

paternal, maternal	data.frames (or objects coercible to data.frames) containing the segments to be shown on the paternal and maternal strands of the karyogram. The first three columns must contain chromosome (with or without "chr" prefix), start position and stop position (in Mb). Column names are ignored, as well as any further columns.
chrom	The (autosomal) chromosomes to be included in the plot, given as a subset of the integers 1, 2,..., 22.
col	A vector of two colours (in any form recognisable by R). If only one colour is given it is recycled. If the vector is named, a colour legend is included in the plot, using the names as labels.
alpha	A single numeric in [0,1] indicating colour transparency.
bgcol	The background colour of the chromosomes.
title	Plot title.

Value

The plot object is returned invisibly, so that additional ggplot layers may be added if needed.

Examples

```
## Not run:
pat = data.frame(chrom = c(1,4,5,5,10,10), start = c(100,50,20,80,10,80),
                 end = c(120,100,25,100,70,120))
mat = data.frame(chrom = c(2,4,5,5,10), start = c(80,50,10,80,50),
                 end = c(120,100,35,100,120))
karyoDiploid(pat, mat)

## End(Not run)
```

---

karyogram2 *Karyogram plots*

Description

Functions for visualising IBD segments in karyograms. The karyogram1() and karyogram2() functions produces karyograms illustrating the output of ibdsim() for one or two specified individuals. The actual plotting is done by functions karyoHaploid() and karyoDiploid().

Usage

```
karyogram2(sim, ids = NULL, verbose = TRUE, ...)
```

Arguments

sim	A genomeSim object.
ids	A vector of one or two ID labels.
verbose	A logical.
...	Further arguments passed on to karyoHaploid().

Value

A plot object returned invisibly.

Examples

```
x = quadHalfFirstCousins()
s = ibdsim(x, seed = 1729)
# karyogram2(s, ids = leaves(x), title = "QHFC")
```

---

karyoHaploid *Haploid karyogram*

Description

Show chromosomal segments in a haploid karyogram

Usage

```
karyoHaploid(
  segments,
  chrom = 1:22,
  colBy = NULL,
  col = NULL,
  separate = TRUE,
  alpha = 1,
  bgcol = "gray92",
  title = NULL,
  legendTitle = NULL,
  base_size = 16
)
```

Arguments

segments	A data.frame (or an object coercible to data.frame) containing the segments to be shown on the karyogram. The first three columns must contain chromosome (with or without "chr" prefix), start position and stop position (in Mb). Any further columns are ignored, except possibly a column indicated by colBy.

chrom	A vector indicating which chromosomes to include.
colBy	A character vector naming the columns to be used for colouring. If NULL (default), all segments have the same colour.
col	A single fill colour for all the segments, or (if colBy is used) a named vector of colours. In the latter case, the names should include all entries in the colBy column.
separate	A logical; relevant only if the colBy column has more than one level. If FALSE, all segments are drawn in full height. This may not be optimal if segments of different colours overlap. If TRUE the levels are drawn in separate bands on the chromosomes.
alpha	A single numeric in [0,1] indicating colour transparency.
bgcol	The background colour of the chromosomes.
title	Plot title.
legendTitle	Legend title.
base_size	Font size, passed to ggplot2::theme().

Value

A ggplot object. Additional ggplot layers may be added if needed.

Examples

```
## Not run:
segs = data.frame(chrom = c(1,4,5,5,10,10),
                  start = c(100,50,20,80,10,50),
                  end = c(120,100,25,100,70,120),
                  IBD = c("paternal","maternal"))
cols = c(paternal = "blue", maternal = "red")

karyoHaploid(segs, col = "cyan")
karyoHaploid(segs, colBy = "IBD", col = cols)

# Note difference if `separate = FALSE`
karyoHaploid(segs, colBy = "IBD", col = cols, separate = FALSE)

# Reduce alpha to see the overlaps:
karyoHaploid(segs, colBy = "IBD", col = cols, separate = FALSE, alpha = 0.7)


## End(Not run)
```

launchApp *Launch the ibdsim2 app*

Description

This launches the Shiny app for simulating IBD segment distributions.

Usage

```
launchApp()
```

Details

The app uses several suggested packages which are not installed by default. To install them together with `ibdsim2`, use `install.packages("ibdsim2", dependencies = TRUE)`.

Value

No return value, called for side effects.

Examples

```
## Not run:
launchApp()

## End(Not run)
```

legacy_decode19 *Legacy version of the* `decode19` *map*

Description

A legacy version of the built-in human recombination map, based on Halldorsson et al., 2019. The implementation of this map used in ibdsim2 was updated in v2.3.0, adding the physical endpoint of each chromosome (this was previously lacking), and using a better thinning algorithm to reduce the raw data given by Halldorsson et al. (2019). See also `loadMap()`.

Usage

```
legacy_decode19
```

Format

A genomeMap object: a list of 23 chromMap objects. Each chromMap is a list containing two numeric matrices, named male and female, and carries these attributes:

- physStart - first physical position (Mb) on the chromosome
- physEnd - last physical position (Mb) on the chromosome
- physRange - physical length, physEnd - physStart (Mb)
- mapLen - length-2 numeric: centiMorgan length of male and female strands
- chrom - chromosome label
- Xchrom - logical flag used by simulators for X-inheritance

Examples

```
loadMap("decode19")         # new
loadMap("legacy_decode19")  # legacy
```

---

loadMap *Load a built-in genetic map*

Description

This function loads one of the built-in genetic maps. Currently, the only option is the fine-scale human recombination map published by Halldorsson et al. (2019). For reasons of speed and efficiency, the map has been thinned down to ~14,000 data points.

Usage

```
loadMap(map = "decode19", chrom = 1:22, uniform = FALSE, sexAverage = FALSE)
```

Arguments

map	The name of the wanted map, possibly abbreviated. Default: "decode19".
chrom	A vector containing a subset of the numbers 1,2,...,23, indicating which chromosomes to load. As a special case, chrom = "X" is synonymous to chrom = 23. Default: 1:22 (the autosomes).
uniform	A logical. If FALSE (default), the complete inhomogeneous map is used. If TRUE, a uniform version of the same map is produced, i.e., with the correct physical range and genetic lengths, but with constant recombination rates along each chromosome.
sexAverage	A logical, by default FALSE. If TRUE, a sex-averaged map is returned, with equal recombination rates for males and females.

Details

NOTE: The built-in map was updated in version 2.3.0, adding more accurate physical chromosome endpoints. While still based on Halldorsson et al. (2019), the new version also uses a better thinning algorithm, allowing to reduce the number of data points from ~38,000 to ~14,000 without losing accuracy. The old version is available as a separate dataset `legacy_decode19` for backwards reproducibility.

By setting `uniform = TRUE`, a uniform version of the map is returned, in which each chromosome has the same genetic lengths as in the original, but with constant recombination rates. This gives much faster simulations and may be preferable in some applications.

Value

An object of class `genomeMap`, which is a list of `chromMap` objects. A `chromMap` is a list of two matrices, named "male" and "female", with various attributes:

- `physStart`: The first physical position (Mb) on the chromosome covered by the map
- `physEnd`: The last physical position (Mb) on the chromosome covered by the map
- `physRange`: The physical map length (Mb), equal to `physEnd - physStart`
- `mapLen`: A vector of length 2, containing the centiMorgan lengths of the male and female strands
- `chrom`: A chromosome label
- `Xchrom`: A logical. This is checked by `ibdsim()` and other function, to select mode of inheritance

References

Halldorsson et al. *Characterizing mutagenic effects of recombination through a sequence-level genetic map.* Science (2019).

See Also

`uniformMap()`, `customMap()`

Examples

```
# By default, the complete map of all 22 autosomes is returned
loadMap()

# Uniform version
m = loadMap(uniform = TRUE)
m

# Check chromosome 1
m1 = m[[1]]
m1
m1$male
m1$female
```

```
# The X chromosome
loadMap(chrom = "X")[[1]]
```

---

maplengths *Physical and genetic map lengths*

Description

Utility functions for extracting the physical or genetic length of chromosome maps and genome maps.

Usage

```
mapLen(x, ...)

## S3 method for class 'chromMap'
mapLen(x, sex = c("male", "female"), ...)

## S3 method for class 'genomeMap'
mapLen(x, sex = c("male", "female"), ...)

physRange(x, ...)

## S3 method for class 'chromMap'
physRange(x, ...)

## S3 method for class 'genomeMap'
physRange(x, ...)
```

Arguments

x	A chromMap or genomeMap object.
...	Not used.
sex	Either "male", "female" or both.

Value

mapLen() returns a numeric of the same length as sex, with the genetic length(s) in centiMorgan.

physRange() returns the physical length (in Mb) of the chromosome/genome covered by the map. For example, for a chromosome map starting at 2 Mb and ending at 8 Mb, the output is 6.

See Also

loadMap(), uniformMap()

Examples

```
m = loadMap(chrom = 1:2)
m

# Applied to `genomeMap` object:
physRange(m)
mapLen(m)

# Applied to `chromMap` object:
physRange(m[[1]])
mapLen(m[[1]])
```

`plotSegmentDistribution`

Scatter plots of IBD segment distributions

Description

Visualise and compare count/length distributions of IBD segments. Two types are currently implemented: Segments of autozygosity (for a single person) and segments with (pairwise) IBD state 1.

Usage

```
plotSegmentDistribution(
  ...,
  type = c("autozygosity", "ibd1"),
  ids = NULL,
  unit = "cm",
  merge = TRUE,
  labels = NULL,
  col = NULL,
  shape = 1,
  alpha = 1,
  ellipses = TRUE,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  legendInside = TRUE
)
```

Arguments

`...`	One or several objects of class `genomeSimList`, typically created by `ibdsim()`. They can be entered separately or as a `list`.
`type`	A string indicating which segments should be plotted. Currently, the allowed entries are "autozygosity" and "ibd1".

ids	A list of the same length as ..., where each entry contains one or two ID labels (depending on type). By default (NULL), these labels are extracted from the inputs in .... Two other short-cuts are possible: If a single vector is given, it is repeated for all pedigrees. Finally, if ids is the word "leaves" then pedtools::leaves() is used to extract labels in each pedigree.
unit	Length unit; either "cm" (centiMorgan) or "mb" (megabases).
merge	A logical, by default TRUE, indicating if adjacent IBD segments should be merged and treated as one.
labels	An optional character vector of labels used in the legend. If NULL, the labels are taken from names(...).
col	An optional colour vector of the same length as ....
shape	A vector with point shapes, of the same length as ....
alpha	A transparency parameter for the scatter points.
ellipses	A logical: Should confidence ellipses be added to the plot? (This requires the MASS package to be installed.)
title, xlab, ylab	Title and axis labels.
legendInside	A logical controlling the legend placement.

Details

This function takes as input one or several complete outputs from the ibdsim(), and produces a scatter plot of the number and average length of IBD segments from each.

Contour curves are added to plot, corresponding to the theoretical/pedigree-based values: either inbreeding coefficients (if type = "autozygosity") or κ_1 (if type = "ibd1").

Value

A ggplot object.

Examples

```
# Simulation parameters used in the below examples.
map = uniformMap(M = 10)   # recombination map
N = 10                     # number of sims

# For more realistic results, replace with e.g.:
# map = loadMap("decode19")
# N = 1000


######################################################################
# EXAMPLE 1
# Comparison of IBD segment distributions
# between paternal and maternal half siblings.
######################################################################
```

```
# Define the pedigrees
xPat = halfSibPed()
xMat = swapSex(xPat, 1)

simPat = ibdsim(xPat, N = N, map = map)
simMat = ibdsim(xMat, N = N, map = map)

# By default, the IBD segments of the "leaves" are computed and plotted
plotSegmentDistribution(simPat, simMat, type = "ibd1", ids = 4:5,
                        labels = c("HSpat", "HSmat"))

######################################################################
# EXAMPLE 2
# Half siblings vs half uncle vs grandparent/grandchild
######################################################################

# Only one pedigree needed here
x = addSon(halfSibPed(), 5)

s = ibdsim(x, N = N, map = map)

# Indicate the pairs explicitly this time.
ids = list(GR = c(2,7), HS = 4:5, HU = c(4,7))

# List names are used as labels in the plot
plotSegmentDistribution(s, type = "ibd1", ids = ids, shape = 1:3)


######################################################################
# EXAMPLE 3
# Comparison of autozygosity distributions in various individuals
# with the same expected inbreeding coefficient (f = 1/8)
######################################################################

G = linearPed(2, sex = 1:2) |> addSon(c(1,5))   # grandfather/granddaughter
HSpat = halfSibPed(sex2 = 2) |> addSon(4:5)     # paternal half sibs
HSmat = swapSex(HSpat, 2)              # maternal half sibs
QHFC = quadHalfFirstCousins()          # quad half first cousins
QHFC = addSon(QHFC, 9:10)

peds = list(G = G, HSpat = HSpat, HSmat = HSmat, QHFC = QHFC)
plotPedList(peds, newdev = TRUE)
dev.off()

# Simulations
s = lapply(peds, function(p)
  ibdsim(p, N = N, ids = leaves(p), verbose = FALSE, map = map))

# Plot distributions
plotSegmentDistribution(s, type = "autoz", title = "Autozygous segments")
```

`profileSimIBD` *Simulate markers conditional on a given IBD pattern*

Description

This function simulates genotypes for a set of markers conditional on a specific underlying IBD pattern, typically produced with `ibdsim()`.

Usage

```
profileSimIBD(
  x,
  ibdpattern,
  ids = NULL,
  markers = NULL,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`x`	A ped object.
`ibdpattern`	A `genomeSim` object, or a list of such objects.
`ids`	A vector of ID labels. If NULL, extracted from `ibdpattern`.
`markers`	A vector with names or indices of markers attached to `x`.
`seed`	An integer seed for the random number generator.
`verbose`	A logical, by default TRUE.

Details

Founder alleles are sampled independently at each marker, so linkage disequilibrium is ignored. Existing genotypes and mutation models are ignored. Individuals not included in `ids` have missing genotypes in the result.

It should be noted that the only *random* part of this function is the sampling of founder alleles. Given those, all other genotypes in the pedigree are determined by the underlying IBD pattern and the marker positions.

Value

A copy of `x` with simulated genotypes, or a list of such copies if `ibdpattern` is a list.

See Also

`ibdsim()`, `forrel::profileSim()`.

Examples

```
# Brother-sister pedigree
ped = nuclearPed(2, sex = 1:2)

# Alleles
als = letters[1:10]


### Autosomal simulation

x = ped |>
  addMarker(alleles = als, chrom = 1, posMb = 20) |>
  addMarker(alleles = als, chrom = 1, posMb = 50) |>
  addMarker(alleles = als, chrom = 1, posMb = 70)

# Simulate the underlying IBD pattern in the pedigree
sim = ibdsim(x, map = uniformMap(M = 1, chrom = 1), seed = 123)

# Simulate genotypes for the sibs conditional on the given IBD pattern
profileSimIBD(x, sim, ids = 3:4, seed = 123)

# With a different seed
profileSimIBD(x, sim, ids = 3:4, seed = 124)


### X chromosomal simulation

y = ped |>
  addMarker(alleles = als, chrom = "X", posMb = 1) |>
  addMarker(alleles = als, chrom = "X", posMb = 50) |>
  addMarker(alleles = als, chrom = "X", posMb = 100)

simy = ibdsim(y, map = loadMap("decode19", chrom = 23), seed = 11)

profileSimIBD(y, simy, seed = 12)
```

`realised` *Realised relatedness*

Description

Compute the realised values of various pedigree coefficients, from simulated data. The current implementation covers inbreeding coefficients for single pedigree members, and kinship, kappa and condensed identity coefficients for pairwise relationships.

Usage

```
realisedInbreeding(
```

```
  sims,
  id = NULL,
  unit = "cm",
  merge = TRUE,
  simplify1 = TRUE
)

realisedKinship(sims, ids = NULL, unit = "cm", merge = TRUE, simplify1 = TRUE)

realisedKappa(sims, ids = NULL, unit = "cm", merge = TRUE, simplify1 = TRUE)

realisedIdentity(sims, ids = NULL, unit = "cm", merge = TRUE, simplify1 = TRUE)
```

Arguments

`sims`	A list of genome simulations, as output by `ibdsim()`.
`id`, `ids`	A vector with one or two ID labels.
`unit`	Either "mb" (megabases) or "cm" (centiMorgan); the length unit for genomic segments. Default is "cm", which normally gives lower variance.
`merge`	A logical, by default TRUE, indicating if adjacent IBD segments should be merged before calculating summary statistics.
`simplify1`	A logical, by default TRUE, simplifying the output if `sims` contains a single simulation. If FALSE, the output is always a list.

Details

The inbreeding coefficient f of a pedigree member is defined as the probability of autozygosity (homozygous for alleles that are identical by descent) in a random autosomal locus. Equivalently, the inbreeding coefficient is the *expected* autozygous proportion of the autosomal chromosomes.

The *realised* inbreeding coefficient f_R in a given individual is the actual fraction of the autosomes covered by autozygous segments. Because of the stochastic nature of meiotic recombination, this may deviate substantially from the pedigree-based expectation.

Similarly, the pedigree-based IBD coefficients $\kappa_0, \kappa_1, \kappa_2$ of noninbred pairs of individuals have realised counterparts. For any given pair of individuals we define k_i to be the actual fraction of the autosome where the individuals share exactly i alleles IBD, where $i = 0, 1, 2$.

Finally, we can do the same thing for each of the nine condensed identity coefficients of Jacquard. For each $i = 1, ..., 9$ we define D_i to be the fraction of the autosome where a given pair of individuals are in identity state i. This uses the conventional ordering of the nine condensed identity states; see for instance the `ribd` GitHub page.

Value

With one simulation and `simplify1 = TRUE`, the realised coefficient(s). Otherwise, a list containing the per-simulation estimates, their mean and their standard deviation.

Examples

```
# Realised IBD coefficients between full siblings
x = nuclearPed(2)
s = ibdsim(x, N = 2) # increase N
realisedKappa(s, ids = 3:4)

############

# Realised inbreeding coefficients, child of first cousins
x = cousinPed(1, child = TRUE)
s = ibdsim(x, N = 2) # increase N
realisedInbreeding(s, id = 9)

# Same data: realised kinship coefficients between the parents
realisedKinship(s, ids = parents(x, 9))

############

# Realised identity coefficients after full sib mating
x = fullSibMating(1)
s = ibdsim(x, N = 2) # increase N
realisedIdentity(s, ids = 5:6)
```

---

segmentStats *Summary statistics for identified segments*

Description

Compute summary statistics for segments identified by `findPattern()`.

Usage

```
segmentStats(
  x,
  quantiles = c(0.025, 0.5, 0.975),
  returnAll = FALSE,
  unit = "mb"
)
```

Arguments

`x`	A list of matrices produced with `findPattern()`.
`quantiles`	A vector of quantiles to include in the summary.
`returnAll`	A logical, by default FALSE. If TRUE, the output includes a vector `allSegs` containing the lengths of all segments in all simulations.
`unit`	Either "mb" (megabases) or "cm" (centiMorgan); the length unit for genomic segments.

Value

A list containing a data frame `perSim`, a matrix `summary` and (if `returnAll` is TRUE) a vector `allSegs`.

Variables used in the output:

- `Count`: The total number of segments in a simulation
- `Total`: The total sum of the segment lengths in a simulation
- `Average`: The average segment lengths in a simulation
- `Shortest`: The length of the shortest segment in a simulation
- `Longest`: The length of the longest segment in a simulation
- `Overall` (only in `summary`): A summary of all segments from all simulations

See Also

`findPattern()`

Examples

```
x = nuclearPed(3)
sims = ibdsim(x, N = 2, map = uniformMap(M = 2), model = "haldane", seed = 1729)

# Segments where all siblings carry the same allele
segs = findPattern(sims, pattern = list(carriers = 3:5))

# Summarise
segmentStats(segs, unit = "mb")

# The unit does not matter in this case (since the map is trivial)
segmentStats(segs, unit = "cm")
```

---

uniformMap *Uniform recombination maps*

Description

Create a uniform recombination map of a given length.

Usage

```
uniformMap(Mb = NULL, cM = NULL, M = NULL, cmPerMb = 1, chrom = 1)
```

Arguments

Mb	Map length in megabases.
cM	Map length in centiMorgan.
M	Map length in Morgan.
cmPerMb	A positive number; the cM/Mb ratio.
chrom	A chromosome label, which may be any string. The values "X" and "23" have a special meaning, both resulting in the Xchrom attribute being set to TRUE.

Value

An object of class chromMap. See loadMap() for details.

See Also

loadMap(), customMap()

Examples

```
m = uniformMap(Mb = 1, cM = 2:3)
m
m$male
m$female

mx = uniformMap(M = 1, chrom = "X")
mx
mx$male
mx$female
```

---

zeroIBD *Probability of zero IBD*

Description

Estimate the probability of no IBD sharing in a pairwise relationship.

Usage

```
zeroIBD(sims, ids = NULL, threshold = 0, unit = "cm")
```

Arguments

sims	A list of genome simulations, as output by ibdsim().
ids	A vector with two ID labels. If NULL (default), these are deduced from the sims object.
threshold	A nonnegative number (default:0). Only IBD segments at least this long are included in the computation.
unit	The unit of measurement for threshold: Either "mb" or "cm" (default).

Value

A list with the following two entries:

- zeroprob: The fraction of sims in which ids have no IBD sharing
- stErr: The standard error of zeroprob

Examples

```
###
# The following example computes the probability of
# no IBD sharing between a pair of fourth cousins.
# We also show how the probability is affected by
# truncation, i.e., ignoring short segments.
###

# Define the pedigree
x = cousinPed(4)
cous = leaves(x)

# Simulate (increase N!)
s = ibdsim(x, N = 10)

# Probability of zero ibd segments. (By default all segs are used)
zeroIBD(s, ids = cous)

# Re-compute with nonzero threshold
zeroIBD(s, ids = cous, threshold = 1, unit = "cm")
zeroIBD(s, ids = cous, threshold = 1, unit = "mb")
```

Index

∗ **datasets**
    legacy_decode19, 18

convertPos, 2
customMap, 3
customMap(), *20*, *30*

estimateCoeffs, 4
estimateIdentity *(*estimateCoeffs*)*, 4
estimateInbreeding *(*estimateCoeffs*)*, 4
estimateKappa *(*estimateCoeffs*)*, 4
estimateKinship *(*estimateCoeffs*)*, 4
estimateTwoLocusIdentity
        *(*estimateCoeffs*)*, 4
estimateTwoLocusInbreeding
        *(*estimateCoeffs*)*, 4
estimateTwoLocusKappa *(*estimateCoeffs*)*,
        4
estimateTwoLocusKinship
        *(*estimateCoeffs*)*, 4
extractIds, 8

findPattern, 8
findPattern(), *28*, *29*

haploDraw, 10

ibdsim, 12
ibdsim(), *3*, *6*, *8–10*, *15*, *22*, *23*, *25*, *27*, *30*

karyoDiploid, 14
karyogram2, 15
karyoHaploid, 16

launchApp, 18
legacy_decode19, 18
loadMap, 19
loadMap(), *4*, *12*, *13*, *18*, *21*, *30*

mapLen *(*maplengths*)*, 21
maplengths, 21

pedtools::founders(), *12*
pedtools::leaves(), *12*
pedtools::ped(), *6*, *12*
pedtools::setFounderInbreeding(), *13*
physRange *(*maplengths*)*, 21
plotSegmentDistribution, 22
profileSimIBD, 25

realised, 26
realisedIdentity *(*realised*)*, 26
realisedInbreeding *(*realised*)*, 26
realisedKappa *(*realised*)*, 26
realisedKinship *(*realised*)*, 26

segmentStats, 28
segmentStats(), *9*
set.seed(), *13*

uniformMap, 29
uniformMap(), *4*, *12*, *13*, *20*, *21*

zeroIBD, 30