Package 'comapr'

November 12, 2026

Title Crossover analysis and genetic map construction

Version 1.17.0

Description comapr detects crossover intervals for single gametes from their haplotype states sequences and stores the crossovers in GRanges object. The genetic distances can then be calculated via the mapping functions using estimated crossover rates for maker intervals. Visualisation functions for plotting interval-based genetic map or cumulative genetic distances are implemented, which help reveal the variation of crossovers landscapes across the genome and across individuals.

biocViews Software, SingleCell, Visualization, Genetics

Depends R (>= 4.1.0)

Imports methods, ggplot2, reshape2, dplyr, gridExtra, plotly, circlize, rlang, GenomicRanges, IRanges, foreach, BiocParallel, GenomeInfoDb, scales, RColorBrewer, tidyr, S4Vectors, utils, Matrix, grid, stats, SummarizedExperiment, plyr, Gviz

License MIT + file LICENSE

Encoding UTF-8

LazyData false

RoxygenNote 7.2.3

VignetteBuilder knitr

Suggests BiocStyle, knitr, rmarkdown, testthat (>= 2.1.0), statmod

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

git_branch devel

git_last_commit b210ba9

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-11-12

Author Ruqian Lyu [aut, cre] (ORCID: <https://orcid.org/0000-0002-7736-6612>)

Maintainer Ruqian Lyu <xiaoru.best@gmail.com>

Contents

.change_missing . . . 2
.filterCOsExtra . . . 3
.label_gt . . . 4
bootstrapDist . . . 5
calGeneticDist . . . 6
coCount . . . 8
comapr . . . 9
combineHapState . . . 9
correctGT . . . 10
countBinState . . . 11
countCOs . . . 12
countGT . . . 13
fill_fail . . . 14
filterGT . . . 14
findDupSamples . . . 15
getAFTracks . . . 16
getCellAFTrack . . . 17
getCellCORange . . . 19
getCellDPTrack . . . 20
getDistortedMarkers . . . 21
getMeanDPTrack . . . 22
getSNPDensityTrack . . . 23
parents_geno . . . 24
perCellChrQC . . . 24
permuteDist . . . 25
perSegChrQC . . . 26
plotCount . . . 27
plotGeneticDist . . . 29
plotGTFreq . . . 30
plotWholeGenome . . . 30
readColMM . . . 31
readHapState . . . 32
snp_geno . . . 33
snp_geno_gr . . . 34
twoSamples . . . 35

Index **36**

`.change_missing` *change SNPs with genotype 'Fail' to* `NA`

Description

change SNPs with genotype 'Fail' to `NA`

Usage

```
.change_missing(s_gt, missing = "Fail")
```

Arguments

s_gt	a column of labelled genotypes
missing	the string used for encoding missing values default to Fail

Details

calculation.

Value

a vector of genotypes with Fail substituted by 'NA'

Author(s)

Ruqian Lyu

.filterCOsExtra *Filter out doublet cells and uninformative SNPs*

Description

This function filter out cells that have been called too many crossovers due to diploid cell contamination or doublets. It also only keeps SNPs (rows) that ever contribute to a crossover interval. This function should be run for individual chromosomes and is called internaly by 'readHapState'

Usage

```
.filterCOsExtra(
  se,
  minSNP = 30,
  minlogllRatio = 200,
  minCellSNP = 200,
  bpDist = 100,
  maxRawCO = 10,
  biasTol = 0.45,
  nmad = 1.5
)
```

Arguments

se	the SummarizedExperiment object that contains the called haplotype state matrix in the assay field and haplotype segment information in the metadata field.
minSNP	the crossover(s) will be filtered out if introduced by a segment that has fewer than 'minSNP' SNPs to support.
minlogllRatio	the crossover(s) will be filtered out if introduced by a segment that has lower than 'minlogllRatio' to its reversed state.
minCellSNP	the minimum number of SNPs detected for a cell to be kept, used with 'nmads'
bpDist	the crossover(s) will be filtered out if introduced by a segment that is shorter than 'bpDist' basepairs.

maxRawCO	if a cell has more than ‘maxRawCO‘ number of raw crossovers called across a chromosome, the cell is filtered out
biasTol	the SNP’s haplotype ratio across all cells is assumed to be 1:1. This argument can be used for removing SNPs that have a biased haplotype. i.e. almost always inferred to be haplotype state 1. It specifies a bias tolerance value, SNPs with haplotype ratios deviating from 0.5 smaller than this value are kept. Only effective when number of cells are larger than 10
nmad	how many mean absolute deviations lower than the median number of SNPs per cellfor a cell to be considered as low coverage cell and filtered Only effective when number of cells are larger than 10. When effective, this or ‘minCellSNP‘, whichever is larger, is applied

Details

The ‘logllRatio‘ value is returned by ‘sgcocaller‘ for each haplotype segment formed by consecutive SNPs that are called to have a same state. It is calculated by taking log of ratio (likelihood of SNPs with inferred states) and (likelihood of SNPs with reversed states)

Value

A ‘RangedSummarizedExperment‘ object that have different dims with input. the colnames are the cell barcodes, rowRanges specify the location of SNPs that contribute to crossovers.

Author(s)

Ruqian Lyu

.label_gt *‘label_gt‘ for changing genotypes in alleles format to labels*

Description

It turns a vector of Genotypes to a vector of Labels consist of ‘Homo_ref‘, ‘Homo_alt‘, and ‘Het‘ given the known genotypes for reference and alternative strains.

Usage

```
.label_gt(s_gt, ref, alt, failed = "Fail")
```

Arguments

s_gt	s_gt, a vector of genotypes for one sample across markers
ref	ref, a vector of genotypes for reference strain across markers
alt	alt, a vector of genotypes for alternative strain across markers
failed	what was used for encoding failed genotype calling such as "Fail" in example

Details

This function takes the a sample's genotype across each SNP marker in alleles and compare with genotypes of in-bred reference and alternative strains to. If the sample's genotype for a particular SNP marker is the same with the reference strain, it is labelled as Homo_ref `homogeneous reference` for a particular SNP marker; if the sample's genotype is the same with the alternative strain it is labelled as Homo_alt `homogeneous alternative` for a particular SNP marker; if the sample's genotype is heterozygous then it is labeled as Het `heterozygous` for this particular genotypes. If it does not fall in any of the three cases, it is labelled as the string specified by the argument 'missing'.

Note that the wrong/failed genotype is labelled as the string in 'missing' after this function. If there is a different label for failed genotype, provide the label using the 'missing' argument.

Value

a vector of labels `Homo_ref, Homo_alt, Het` indicating the progeny's genotypes across markers

Author(s)

Ruqian Lyu

`bootstrapDist` *bootstrapDist*

Description

Generating distribution of sample genetic distances

Usage

```
bootstrapDist(co_gr, B = 1000, mapping_fun = "k", group_by)
```

Arguments

`co_gr`	GRanges or RangedSummarizedExperiment object that contains the crossover counts for each marker interval across all samples. Returned by `countCOs`
`B`	integer the number of sampling times
`mapping_fun`	character default to "k" (kosambi mapping function). It can be one of the mapping functions: "k","h"
`group_by`	the prefix for each group that we need to generate distributions for(only when co_gr is a GRanges object). Or the column name for 'colData(co_gr)' that contains the group factor (only when co_gr is a RangedSummarizedExperiment object)

Details

It takes the crossover counts for samples in multiple groups that is returned by 'countCO'. It then draws samples from a group with replacement and calculate the distribution of relevant statistics.

Value

lists of numeric genetic distances for multiple samples

Author(s)

Ruqian Lyu

Examples

```
data(coCount)

bootsDiff <- bootstrapDist(coCount, group_by = "sampleGroup",B=10)
```

---

calGeneticDist *calGeneticDist*

Description

Calculate genetic distances of marker intervals or binned-chromosome Given whether crossover happens in each marker interval, calculate the recombination fraction in samples and then derive the Haldane or Kosambi genetic distances via mapping functions

Usage

```
calGeneticDist(
  co_count,
  bin_size = NULL,
  mapping_fun = "k",
  ref_genome = "mm10",
  group_by = NULL,
  chrom_info = NULL
)

## S4 method for signature 'GRanges,missing,ANY,ANY,missing'
calGeneticDist(
  co_count,
  bin_size = NULL,
  mapping_fun = "k",
  ref_genome = "mm10",
  group_by = NULL,
  chrom_info = NULL
)

## S4 method for signature 'GRanges,numeric,ANY,ANY,missing'
calGeneticDist(
  co_count,
  bin_size = NULL,
  mapping_fun = "k",
  ref_genome = "mm10",
  group_by = NULL,
  chrom_info = NULL
)

## S4 method for signature 'GRanges,missing,ANY,ANY,character'
calGeneticDist(
```

```
  co_count,
  bin_size = NULL,
  mapping_fun = "k",
  ref_genome = "mm10",
  group_by = NULL,
  chrom_info = NULL
)

## S4 method for signature 'GRanges,numeric,ANY,ANY,character'
calGeneticDist(
  co_count,
  bin_size = NULL,
  mapping_fun = "k",
  ref_genome = "mm10",
  group_by = NULL,
  chrom_info = NULL
)

## S4 method for signature 'RangedSummarizedExperiment,missing,ANY,ANY,missing'
calGeneticDist(
  co_count,
  bin_size = NULL,
  mapping_fun = "k",
  ref_genome = "mm10",
  group_by = NULL,
  chrom_info = NULL
)

## S4 method for signature
## 'RangedSummarizedExperiment,missing,ANY,ANY,character'
calGeneticDist(
  co_count,
  bin_size = NULL,
  mapping_fun = "k",
  ref_genome = "mm10",
  group_by = NULL,
  chrom_info = NULL
)

## S4 method for signature
## 'RangedSummarizedExperiment,numeric,ANY,ANY,character'
calGeneticDist(
  co_count,
  bin_size = NULL,
  mapping_fun = "k",
  ref_genome = "mm10",
  group_by = NULL,
  chrom_info = NULL
)

## S4 method for signature 'RangedSummarizedExperiment,numeric,ANY,ANY,missing'
calGeneticDist(
```

```
  co_count,
  bin_size = NULL,
  mapping_fun = "k",
  ref_genome = "mm10",
  group_by = NULL,
  chrom_info = NULL
)
```

Arguments

co_count	GRange or RangedSummarizedExperiment object, returned by countCO
bin_size	The binning size for grouping marker intervals into bins. If not supplied,the orginial marker intervals are returned with converted genetic distancens based on recombination rate
mapping_fun	The mapping function to use, can be one of "k" or "h" (kosambi or haldane)
ref_genome	The reference genome name. It is used to fetch the chromosome size information from UCSC database.
group_by	character vector contains the unique prefix of sample names that are used for defining different sample groups. Or the column name in colData(co_count) that specify the group factor. If missing all samples are assumed to be from one group
chrom_info	A user supplied data.frame containing two columns with column names chrom and size, describing the chromosome names and lengths if not using ref_genome from UCSC. If supplied, the 'ref_genome' is ignored.

Value

GRanges object GRanges for marker intervals or binned intervals with Haldane or Kosambi centi-Morgans

Examples

```
data(coCount)
dist_se <- calGeneticDist(coCount)
# dist_se <- calGeneticDist(coCount,group_by="sampleGroup")
```

---

coCount *RangedSummarizedExperiment object containing the crossover counts across samples for the list of SNP marker intervals*

Description

RangedSummarizedExperiment object containing the crossover counts across samples for the list of SNP marker intervals

Usage

```
data(coCount)
```

Format

An object of class `RangedSummarizedExperiment` with 3 rows and 10 columns.

`comapr` `comapr` *package*

Description

crossover inference package

Details

See the README on GitLab

`combineHapState` *combineHapState*

Description

combine two ‘RangedSummarizedExperiment‘ objects, each contains the haplotype state for a list of SNPs across a set of cells. The combined result will have cells from two individuals and merged list of SNPs from the two.

Usage

```
combineHapState(rse1, rse2, groupName = c("Sample1", "Sample2"))
```

Arguments

`rse1`	the first ‘RangedSummarizedExperiment‘
`rse2`	the second ‘RangedSummarizedExperiment‘
`groupName`	a character vector of length 2 that contains the first and the second group’s names

Value

A ‘RangedSummarizedExperiment‘ that contains the cells and SNPs in both ‘rse‘

Author(s)

Ruqian Lyu

Examples

```
BiocParallel::register(BiocParallel::SnowParam(workers = 1))
demo_path <- paste0(system.file("extdata",package = "comapr"),"/")
s1_rse_state <- readHapState("s1",chroms=c("chr1"),
                             path=demo_path,barcodeFile=NULL,minSNP = 0,
                             minlogllRatio = 50,
                             bpDist = 100,maxRawCO=10,
                             minCellSNP = 1)

s2_rse_state <- readHapState("s2",chroms=c("chr1"),
                             path=demo_path,
                             barcodeFile=paste0(demo_path,"s2_barcodes.txt"),
                             minSNP = 0,
                             minlogllRatio = 50,
                             bpDist = 100,maxRawCO=10,
                             minCellSNP = 1)
sb <- combineHapState(s1_rse_state,s2_rse_state)
```

`correctGT` *correctGT*

Description

function for formatting and correction genotypes of markers

Usage

```
correctGT(gt_matrix, ref, alt, failed = "Fail", wrong_label = "Homo_ref")
```

Arguments

`gt_matrix`	the input genotype matrix of markers by samples with rownames as marker IDs and column names as sample IDs
`ref`	a vector of genotypes of the inbred reference strain
`alt`	a vector of genotypes of the inbred alternative strain
`failed`	what was used for encoding failed genotype calling such as "Fail" in example data `snp_geno`
`wrong_label`	what would be considered a wrong genotype label for example Homo_ref which should not be in the possible genotypes of BC1F1 samples

Details

This function changes genotype in alleles to genotype labels, change Homo_ref to Hets/Fail, infer Failed genotype, and change "Failed" to NA for counting crossover later

This function changes genotype in alleles to labels by calling internal functions `lable_gt`, and changes the wrong genotype Homo_ref to Fail by calling `.change_missing`.

Value

a genotype data.frame of sample genotypes with dimension as the input ‘gt_matrix‘ with genotypes converted to labels and failed calls are changed to NA.

Author(s)

Ruqian Lyu

Examples

```
data(snp_geno_gr)
data(parents_geno)
snp_gt_crt <- correctGT(gt_matrix = GenomicRanges::mcols(snp_geno_gr),
                        ref = parents_geno$ref,
                        alt = parents_geno$alt,
                        fail = "Fail",
                        wrong_label = "Homo_ref")
```

`countBinState` *countBinState*

Description

Bins the chromosome into supplied number of bins and find the state of the chromosome bins across all gamete cells

Usage

```
countBinState(chr, snpAnno, viState, genomeRange, ntile = 5)
```

Arguments

`chr`	character, the chromosome to check
`snpAnno`	data.frame, the SNP annotation for the supplied chromosome
`viState`	dgTMatrix/Matrix, the viterbi state matrix, output from ‘sgcocaller‘
`genomeRange`	GRanges object with seqlengths information for the genome
`ntile`	integer, how many tiles the chromosome is binned into

Details

This function is used for checking whether chromosome segregation pattern obeys the expected ratio.

Value

a data.frame that contains chromosome bin segregation ratio

Author(s)

Ruqian Lyu

Examples

```
library(IRanges)
library(S4Vectors)

chrom_info <- GenomeInfoDb::getChromInfoFromUCSC("mm10")
seq_length <- chrom_info$size
names(seq_length) <- chrom_info$chrom

dna_mm10_gr <- GenomicRanges::GRanges(
  seqnames = Rle(names(seq_length)),
  ranges = IRanges(1, end = seq_length, names = names(seq_length)),
  seqlengths = seq_length)

GenomeInfoDb::genome(dna_mm10_gr) <- "mm10"
demo_path <- system.file("extdata",package = "comapr")
sampleName <- "s1"
chr <- "chr1"
vi_mtx <- Matrix::readMM(file = paste0(demo_path,"/", sampleName, "_",
                                       chr, "_vi.mtx"))

snpAnno <- read.table(file = paste0(demo_path,"/", sampleName,
                                    "_", chr, "_snpAnnot.txt"),
                                    stringsAsFactors = FALSE,
                      header = TRUE)

countBinState(chr = "chr1",snpAnno = snpAnno,
viState = vi_mtx,genomeRange = dna_mm10_gr, ntile = 1)
```

---

countCOs *countCOs*

Description

Count number of COs within each marker interval COs identified in the interval overlapping missing markers are distributed according to marker interval base-pair sizes. Genotypes encoded with "0" are treated as missing value.

Usage

```
countCOs(geno)

## S4 method for signature 'GRanges'
countCOs(geno)

## S4 method for signature 'RangedSummarizedExperiment'
countCOs(geno)
```

Arguments

geno	GRanges object or RangedSummarizedExperiment object with genotype matrix that has SNP positions in the rows and cells/samples in the columns

Value

GRanges object or RangedSummarizedExperiment with markers-intervals as rows and samples in columns, values as the number of COs estimated for each marker interval

Author(s)

Ruqian Lyu

Examples

```
data(twoSamples)
cocount <- countCOs(twoSamples)
```

---

countGT *countGT*

Description

count how many samples have genotypes calls across markers and count how many markers that each individual has called genotypes for. This function helps identify poor samples or poor markers for filtering. It can also generate plots that help identify outlier samples/markers

Usage

```
countGT(geno, plot = TRUE, interactive = FALSE)
```

Arguments

geno	the genotype data.frame of markers by samples from output of function correctGT
plot	it determines whether a plot will be generated, defaults to TRUE
interactive	it determines whether an interactive plot will be generated

Value

A list of two elements including n_markers and n_samples

Author(s)

Ruqian Lyu

Examples

```
data(snp_geno_gr)
genotype_counts <- countGT(GenomicRanges::mcols(snp_geno_gr))
```

`fill_fail` *Infer the genotype of failed SNPs If we have a* `Fail` *in the genotype data and the* `Fail` *in a block of either Home_alt, or Het, we fill in the* `Fail`*s using values of the ones adjacent to it, otherwise they remain as "Fail" to indicate missing values.*

Description

Infer the genotype of failed SNPs If we have a `Fail` in the genotype data and the `Fail` in a block of either Home_alt, or Het, we fill in the `Fail`s using values of the ones adjacent to it, otherwise they remain as "Fail" to indicate missing values.

Usage

```
fill_fail(s_gt, fail = "Fail", chr = NULL)
```

Arguments

`s_gt`	a column of labelled genotypes
`fail`	the string that is used for encoding failed genotype results, default to `Fail`
`chr`	the factor vector indicating which chromosomes the markers are on, default to `NULL` which means the input markers are all on the same chromosome.

Value

a vector of genotypes with Failed genotype imputed or changed to ‘NA‘ if not imputable

Author(s)

Ruqian Lyu

`filterGT` *filterGT*

Description

Filter markers or samples that have too many missing values

Usage

```
filterGT(geno, min_markers = 5, min_samples = 3)

## S4 method for signature 'matrix,numeric,numeric'
filterGT(geno, min_markers = 5, min_samples = 3)

## S4 method for signature 'GRanges,numeric,numeric'
filterGT(geno, min_markers = 5, min_samples = 3)
```

Arguments

geno	the genotype data.frame of markers by samples from output of function correctGT
min_markers	the minimum number of markers for a sample to be kept
min_samples	the minimum number of samples for a marker to be kept

Details

This function takes the geno data.frame and filter the data.frame by the provided cut-offs.

Value

The filtered genotype matrix

Author(s)

Ruqian Lyu

Examples

```
data(snp_geno_gr)
corrected_geno <- filterGT(snp_geno_gr, min_markers = 30,min_samples = 2)
```

---

findDupSamples *findDupSamples*

Description

Find the duplicated samples by look at the number of matching genotypes in all pair-wise samples

Usage

```
findDupSamples(geno, threshold = 0.99, in_text = FALSE)
```

Arguments

geno	the genotype data.frame of markers by samples from output of function correctGT
threshold	the frequency cut-off of number of matching genotypes out of all geneotypes for determining whether the pair of samples are duplicated, defaults to 0.99. NAs are regarded as same genotypes for two samples if they both have NA for a marker.
in_text	whether text of frequencies should be displayed in the heatmap cells

Value

The paris of duplicated samples.

Author(s)

Ruqian Lyu

Examples

```
data(snp_geno)
or_geno <- snp_geno[,grep("X",colnames(snp_geno))]
rownames(or_geno) <- paste0(snp_geno$CHR,"_",snp_geno$POS)
or_geno[,1] <- or_geno[,5]
cr_geno <- correctGT(or_geno,ref = snp_geno$C57BL.6J,
                     alt = snp_geno$FVB.NJ..i.)
dups <- findDupSamples(cr_geno)
```

`getAFTracks` *getAFTracks*

Description

Generate the raw alternative allele frequencies tracks for all cells in the columns of provided ‘co_count‘

Usage

```
getAFTracks(
  chrom = "chr1",
  path_loc = "./output/firstBatch/WC_522/",
  sampleName = "WC_522",
  nwindow = 80,
  barcodeFile,
  co_count,
  snp_track = NULL
)
```

Arguments

`chrom`	the chromosome
`path_loc`	the path prefix to the output files from sscocaller including "*_totalCount.mtx" and "_altCount.mtx"
`sampleName`	the sample name, which is the prefix of sscocaller's output files
`nwindow`	the number of windows for binning the chromosome
`barcodeFile`	the barcode file containing the list of cell barcodes used as the input file for sscocaller
`co_count`	‘GRange‘ or ‘RangedSummarizedExperiment‘ object, returned by `countCO` that contains the crossover intervals and the number of crossovers in each cell.
`snp_track`	the SNP position track which is used for obtaining the SNP chromosome locations. It could be omitted and the SNP positions will be acquired from the "*_snpAnnot.txt" file.

Value

a list object, in which each element is a list of two items with the cell's alternative allele frequency DataTrack and the called crossover ranges.

Author(s)

Ruqian Lyu

Examples

```
demo_path <- system.file("extdata",package = "comapr")
s1_rse_state <- readHapState("s1",chroms=c("chr1"),
                             path=demo_path,barcodeFile=NULL,minSNP = 0,
                             minlogllRatio = 50,
                             bpDist = 100,maxRawCO=10,
                             minCellSNP = 0)
s1_counts <- countCOs(s1_rse_state)

af_co_tracks <- getAFTracks(chrom ="chr1",
                            path_loc = demo_path,
                            sampleName = "s1",
                            barcodeFile = file.path(demo_path,
                                                    "s1_barcodes.txt"),
                            co_count = s1_counts)
```

---

getCellAFTrack *getCellAFTrack Generates the DataTracks for plotting AF and crossover regions*

Description

It plots the raw alternative allele frequencies and highlight the crossover regions for the selected cell.

It plots the raw alternative allele frequencies and highlight the crossover regions for the selected cell.

Usage

```
getCellAFTrack(
  chrom = "chr1",
  path_loc = "./output/firstBatch/WC_522/",
  sampleName = "WC_522",
  nwindow = 80,
  barcodeFile,
  cellBarcode,
  co_count,
  snp_track = NULL,
  chunk = 1000L
)

getCellAFTrack(
  chrom = "chr1",
  path_loc = "./output/firstBatch/WC_522/",
  sampleName = "WC_522",
  nwindow = 80,
```

```
  barcodeFile,
  cellBarcode,
  co_count,
  snp_track = NULL,
  chunk = 1000L
)
```

Arguments

chrom	the chromosome
path_loc	the path prefix to the output files from sscocaller including "*_totalCount.mtx" and "_altCount.mtx"
sampleName	the sample name, which is the prefix of sscocaller's output files
nwindow	the number of windows for binning the chromosome
barcodeFile	the barcode file containing the list of cell barcodes used as the input file for sscocaller
cellBarcode	the selected cell barcode
co_count	'GRange' or 'RangedSummarizedExperiment' object, returned by countCO that contains the crossover intervals and the number of crossovers in each cell.
snp_track	the SNP position track which is used for obtaining the SNP chromosome locations. It could be omitted and the SNP positions will be acquired from the "*_snpAnnot.txt" file.
chunk	An integer scalar indicating the chunk size to use, i.e., number of rows to read at any one time.

Value

The DataTrack object defined in DataTrack

The DataTrack object defined in DataTrack

Author(s)

Ruqian Lyu

Examples

```
demo_path <-paste0(system.file("extdata",package = "comapr"),"/")
s1_rse_state <- readHapState("s1",chroms=c("chr1"),
                             path=demo_path,barcodeFile=NULL,minSNP = 0,
                             minlogllRatio = 50,
                             bpDist = 100,maxRawCO=10,
                             minCellSNP = 0)
s1_counts <- countCOs(s1_rse_state)
af_co_tracks <- getCellAFTrack(chrom ="chr1",
                               path_loc = demo_path,
                               sampleName = "s1",
                               barcodeFile = paste0(demo_path,
                                                    "s1_barcodes.txt"),
                               cellBarcode = "BC1",
                               co_count = s1_counts)

demo_path <-paste0(system.file("extdata",package = "comapr"),"/")
```

```
s1_rse_state <- readHapState("s1",chroms=c("chr1"),
                             path=demo_path,barcodeFile=NULL,minSNP = 0,
                             minlogllRatio = 50,
                             bpDist = 100,maxRawCO=10,
                             minCellSNP = 0)
s1_counts <- countCOs(s1_rse_state)
af_co_tracks <- getCellAFTrack(chrom ="chr1",
                               path_loc = demo_path,
                               sampleName = "s1",
                               barcodeFile = paste0(demo_path,
                                                    "s1_barcodes.txt"),
                               cellBarcode = "BC1",
                               co_count = s1_counts)
```

`getCellCORange` *getCellCORange*

Description

It finds the crossover intervals for a selected cell

Usage

```
getCellCORange(co_count, cellBarcode)
```

Arguments

`co_count`	'GRanges' or 'RangedSummarizedExperiment' object,
`cellBarcode`	the selected cell's barcode

Value

GRange object containing the crossover intervals for the selected cell

Author(s)

Ruqian Lyu

Examples

```
  demo_path <-paste0(system.file("extdata",package = "comapr"),"/")
s1_rse_state <- readHapState("s1",chroms=c("chr1"),
                             path=demo_path,barcodeFile=NULL,minSNP = 0,
                             minlogllRatio = 50,
                             bpDist = 100,maxRawCO=10,
                             minCellSNP = 0)
s1_counts <- countCOs(s1_rse_state)

co_ranges <- getCellCORange(cellBarcode = "BC1",
                            co_count = s1_counts)
```

`getCellDPTrack` *getCellDPTrack Generates the DataTrack for plotting DP of a selected cell*

Description

It plots the total allele counts for the selected cell.

Usage

```
getCellDPTrack(
  chrom = "chr1",
  path_loc = "./output/firstBatch/WC_522/",
  sampleName = "WC_522",
  nwindow = 80,
  barcodeFile,
  cellBarcode,
  snp_track = NULL,
  chunk = 1000L,
  log = TRUE,
  plot_type = "hist"
)
```

Arguments

`chrom`	the chromosome
`path_loc`	the path prefix to the output files from sscocaller including "*_totalCount.mtx"
`sampleName`	the sample name, which is the prefix of sscocaller's output files
`nwindow`	the number of windows for binning the chromosome
`barcodeFile`	the barcode file containing the list of cell barcodes used as the input file for sscocaller
`cellBarcode`	the selected cell barcode
`snp_track`	the SNP position track which is used for obtaining the SNP chromosome locations. It could be omitted and the SNP positions will be acquired from the "*_snpAnnot.txt" file.
`chunk`	A integer scalar indicating the chunk size to use, i.e., number of rows to read at any one time.
`log`	whether the histogram of SNP density should be plotted on log scale (log10)
`plot_type`	the DataTrack plot type, default to be 'hist'

Value

The DataTrack object defined in `DataTrack`

Author(s)

Ruqian Lyu

Examples

```
demo_path <- system.file("extdata",package = "comapr")
s1_rse_state <- readHapState("s1",chroms=c("chr1"),
                             path=demo_path,barcodeFile=NULL,minSNP = 0,
                             minlogllRatio = 50,
                             bpDist = 100,maxRawCO=10,
                             minCellSNP = 0)
s1_counts <- countCOs(s1_rse_state)
dp_co_tracks <- getCellDPTrack(chrom ="chr1",
                               path_loc = demo_path,
                               sampleName = "s1",
                               barcodeFile = file.path(demo_path,
                                                       "s1_barcodes.txt"),
                               cellBarcode = "BC1")
```

`getDistortedMarkers` *getDistortedMarkers*

Description

Marker segregation distortion detection using chisq-test

Usage

```
getDistortedMarkers(geno, p = c(0.5, 0.5), adj.method = "BH")
```

Arguments

`geno`	the genotype data.frame of markers by samples from output of function `correctGT`
`p`	the expected geneotype ratio in a numeric vector, defaults to c(0.5,0.5)
`adj.method`	Methods to adjust for multiple comparisons, defaults to "BH"

Details

We expect the genotypes to appear with the frequencies of 1:1 homo_alt:hets. We usechisq.test for finding markers that have genotypes among samples that are significantly different from the 1:1 ratio and report them

Value

data.frame with each row representing one SNP marker and columns containing the chisq.test results

Author(s)

Ruqian Lyu

Examples

```
data(parents_geno)
data(snp_geno_gr)
corrected_geno <- correctGT(gt_matrix = GenomicRanges::mcols(snp_geno_gr),
ref = parents_geno$ref,alt = parents_geno$alt,fail = "Fail",
wrong_label = "Homo_ref")
GenomicRanges::mcols(snp_geno_gr) <- corrected_geno
corrected_geno <- filterGT(snp_geno_gr, min_markers = 30,min_samples = 2)
pvalues <- getDistortedMarkers(GenomicRanges::mcols(corrected_geno))
```

`getMeanDPTrack` *getMeanDPTrack*

Description

Generate the mean DP (Depth) DataTrack (from Gviz) for cells

Usage

```
getMeanDPTrack(
  chrom = "chr1",
  path_loc,
  nwindow = 80,
  sampleName,
  barcodeFile,
  plot_type = "hist",
  selectedBarcodes = NULL,
  snp_track = NULL,
  log = TRUE
)
```

Arguments

`chrom`	the chromosome
`path_loc`	the path prefix to the output files from sscocaller including "*_totalCount.mtx" and "_altCount.mtx"
`nwindow`	the number of windows for binning the chromosome
`sampleName`	the sample name, which is the prefix of sscocaller's output files
`barcodeFile`	the barcode file containing the list of cell barcodes used as the input file for sscocaller
`plot_type`	the DataTrack plot type, default to be 'hist'
`selectedBarcodes`	the selected cell barcodes which should be the barcodes that have been called crossovers for. If not supplied then all cells are counted.
`snp_track`	the SNP position track which is used for obtaining the SNP chromosome locations. It could be omitted and the SNP positions will be acquired from the "*_snpAnnot.txt" file.
`log`	whether the histogram of SNP density should be plotted on log scale (log10)

Value

DataTrack object plotting the mean DP histogram for windowed chromosomes

Author(s)

Ruqian Lyu

Examples

```
demo_path <-paste0(system.file("extdata",package = "comapr"),"/")
meanDP_track <- getMeanDPTrack(chrom ="chr1",
                               path_loc = demo_path,
                               sampleName = "s1",
                               barcodeFile = paste0(demo_path,
                                                    "s1_barcodes.txt"))
```

`getSNPDensityTrack` *getSNPDensityTrack*

Description

Generate the SNP density DataTrack (from 'Gviz') for selected chromosome

Usage

```
getSNPDensityTrack(
  chrom = "chr1",
  sampleName = "s1",
  path_loc = ".",
  nwindow = 80,
  plot_type = "hist",
  log = TRUE
)
```

Arguments

`chrom`	the chromosome
`sampleName`	the sample name, which is the prefix of sscocaller's output files
`path_loc`	the path prefix to the output files from sscocaller including "*_totalCount.mtx" and "_altCount.mtx"
`nwindow`	the number of windows for binning the chromosome
`plot_type`	the DataTrack plot type, default to be 'hist'
`log`	whether the histogram of SNP density should be plotted on log scale (log10)

Value

DataTrack object plotting the SNP density histogram

Author(s)

Ruqian Lyu

Examples

```
demo_path <- system.file("extdata",package = "comapr")
snp_track <- getSNPDensityTrack(chrom ="chr1",
                                path_loc = demo_path,
                                sampleName = "s1")
```

`parents_geno` *Parents' genotype for F1 samples in 'snp_geno'*

Description

Parents' genotype for F1 samples in 'snp_geno'

Usage

```
data(parents_geno)
```

Format

A data.frame:

C57BL.6J genotype of reference mouse train across markers

FVB.NJ..i. genotype of alternative mouse train across markers

---

`perCellChrQC` *perCellChrQC*

Description

A function that parses output ('_viSegInfo.txt')from 'sgcocaller' https://gitlab.svi.edu.au/biocellgen-public/sgcocaller and generate cell cell (per chr) summary statistics

Usage

```
perCellChrQC(
  sampleName,
  chroms = c("chr1", "chr7", "chr15"),
  path,
  barcodeFile = NULL,
  doPlot = TRUE
)
```

Arguments

sampleName	the name of the sample to parse which is used as prefix for finding relevant files for the underlying sample
chroms	the character vectors of chromosomes to parse. Multiple chromosomes' results will be concated together.
path	the path to the files, with name patterns *chrom_vi.mtx, *chrom_viSegInfo.txt, end with slash
barcodeFile	defaults to NULL, it is assumed to be in the same d irectory as the other files and with name sampleName_barcodes.txt
doPlot	whether a plot should returned, default to TRUE

Value

a list object that contains the data.frame with summarised statistics per chr per cell and a plot (if doPlot)

Author(s)

Ruqian Lyu

Examples

```
demo_path <-system.file("extdata",package = "comapr")
pcQC <- perCellChrQC(sampleName="s1",chroms=c("chr1"),
path=demo_path,
barcodeFile=NULL)
```

---

permuteDist *permuteDist*

Description

Permutation test of two sample groups

Usage

```
permuteDist(co_gr, B = 100, mapping_fun = "k", group_by)
```

Arguments

co_gr	GRanges or RangedSummarizedExperiment object that contains the crossover counts for each marker interval across all samples. Returned by countCOs
B	integer the number of sampling times
mapping_fun	character default to "k" (kosambi mapping function). It can be one of the mapping functions: "k","h"
group_by	the prefix for each group that we need to generate distributions for(only when co_gr is a GRanges object). Or the column name for 'colData(co_gr)' that contains the group factor (only when co_gr is a RangedSummarizedExperiment object)

Details

It shuffles the group labels for the samples and calculate a difference between two groups after shuffling.

Value

A list of three elements. ‘permutes‘ of length B with numeric differences of permuted group differences,‘observed_diff‘ the observed genetic distances of two groups, ‘nSample‘, the number of samples in the first and second group.

Author(s)

Ruqian Lyu

Examples

```
data(coCount)
perms <- permuteDist(coCount, group_by = "sampleGroup",B=10)
```

`perSegChrQC` *perSegChrQC*

Description

Plots the summary statistics of segments that are generated by ‘sgcocaller‘ https://gitlab.svi.edu.au/biocellgen-public/sgcocaller which have been detected by finding consequtive viter states along the list of SNP markers.

Usage

```
perSegChrQC(
  sampleName,
  chroms = c("chr1", "chr7", "chr15"),
  path,
  barcodeFile = NULL,
  maxRawCO = 10
)
```

Arguments

`sampleName`	the name of the sample to parse which is used as prefix for finding relevant files for the underlying sample
`chroms`	the vector of chromosomes
`path`	the path to the files, with name patterns *chrom_vi.mtx, *chrom_viSegInfo.txt, end with slash
`barcodeFile`	defaults to NULL, it is assumed to be in the same directory as the other files and with name sampleName_barcodes.txt
`maxRawCO`	if a cell has more than ‘maxRawCO‘ number of raw crossovers called across a chromosome, the cell is filtered out#’

Details

It provides guidance in filtering out close double crossovers that are not likely biological but due to technical reasons as well as crossovers that are supported by fewer number of SNPs at the ends of the chromosomes.

Value

Histogram plots for statistics summarized across all Viterbi state segments

Author(s)

Ruqian Lyu

Examples

```
demo_path <- system.file("extdata",package = "comapr")
s1_rse_qc <- perSegChrQC(sampleName="s1",
                         chroms=c("chr1"),
                         path=demo_path, maxRawCO=10)
```

---

plotCount *plotCount*

Description

Plot the number of COs per sample group or per chromosome

Usage

```
plotCount(
  co_count,
  by_chr = FALSE,
  group_by = "sampleGroup",
  plot_type = "error_bar"
)

## S4 method for signature 'RangedSummarizedExperiment,missing,missing'
plotCount(
  co_count,
  by_chr = FALSE,
  group_by = "sampleGroup",
  plot_type = "error_bar"
)

## S4 method for signature 'RangedSummarizedExperiment,missing,character'
plotCount(
  co_count,
  by_chr = FALSE,
  group_by = "sampleGroup",
  plot_type = "error_bar"
```

```
)

## S4 method for signature 'RangedSummarizedExperiment,logical,character'
plotCount(
  co_count,
  by_chr = FALSE,
  group_by = "sampleGroup",
  plot_type = "error_bar"
)

## S4 method for signature 'RangedSummarizedExperiment,logical,missing'
plotCount(
  co_count,
  by_chr = FALSE,
  group_by = "sampleGroup",
  plot_type = "error_bar"
)

## S4 method for signature 'GRanges,logical,missing'
plotCount(
  co_count,
  by_chr = FALSE,
  group_by = "sampleGroup",
  plot_type = "error_bar"
)

## S4 method for signature 'GRanges,missing,missing'
plotCount(
  co_count,
  by_chr = FALSE,
  group_by = "sampleGroup",
  plot_type = "error_bar"
)

## S4 method for signature 'GRanges,missing,character'
plotCount(
  co_count,
  by_chr = FALSE,
  group_by = "sampleGroup",
  plot_type = "error_bar"
)

## S4 method for signature 'GRanges,logical,character'
plotCount(
  co_count,
  by_chr = FALSE,
  group_by = "sampleGroup",
  plot_type = "error_bar"
)
```

Arguments

co_count	GRange or RangedSummarizedExperiment object, returned by countCO

by_chr	whether it should plot each chromosome separately
group_by	the column name in ‘colData(co_count)‘ that specify the grouping factor. Or the character vector contains the unique prefix of sample names that are used for defining different sample groups. If missing all samples are assumed to be from one group
plot_type	determins what type the plot will be, choose from "error_bar" or "hist". Only relevant when by_chr=TRUE

Value

ggplot object

Examples

```
demo_path <-paste0(system.file("extdata",package = "comapr"),"/")
s1_rse_state <- readHapState("s1",chroms=c("chr1"),
                             path=demo_path,barcodeFile=NULL,minSNP = 0,
                             minlogllRatio = 50,
                             bpDist = 100,maxRawCO=10,
                             minCellSNP = 0)
s1_count <- countCOs(s1_rse_state)
plotCount(s1_count)
```

---

plotGeneticDist *plotGeneticDist*

Description

Plotting the calculated genetic distanced for each bin or marker interval supplied by the GRanges object

Usage

```
plotGeneticDist(gr, bin = TRUE, chr = NULL, cumulative = FALSE)
```

Arguments

gr	GRanges object with genetic distances calculated for marker intervals
bin	TRUE or FALSE, indicating whether the supplied GRange objecct is for binned interval
chr	the specific chrs selected to plot
cumulative	TRUE or FALSE, indicating whether it plots the bin-wise genetic distances or the cumulative distances

Value

ggplot2 plot

Author(s)

Ruqian Lyu

Examples

```
data(coCount)
dist_se <- calGeneticDist(coCount)
plotGeneticDist(SummarizedExperiment::rowRanges(dist_se))
```

---

plotGTFreq *plotGTFreq*

Description

Function to plot the genotypes for all samples faceted by genotype

Usage

```
plotGTFreq(geno)
```

Arguments

geno	the genotype data.frame of markers by samples from output of function correctGT

Value

A ggplot object

Author(s)

Ruqian Lyu

Examples

```
data(snp_geno)
or_geno <- snp_geno[,grep("X",colnames(snp_geno))]
rownames(or_geno) <- paste0(snp_geno$CHR,"_",snp_geno$POS)
or_geno[1,] <- rep("Fail",dim(or_geno)[2])
cr_geno <- correctGT(or_geno,ref = snp_geno$C57BL.6J,
                     alt = snp_geno$FVB.NJ..i.)
ft_gt <- filterGT(cr_geno)
plotGTFreq(ft_gt)
```

---

plotWholeGenome *Plot cumulative genetic distances across the genome*

Description

This function takes the calculated genetic distances for all marker intervals across all chromosomes provided and plot the cumulative genetic distances

Usage

```
plotWholeGenome(gr)
```

Arguments

gr	GRanges object with genetic distances calculated for marker intervals

Value

A ggplot object

Examples

```
data(coCount)
dist_se <- calGeneticDist(coCount)
plotWholeGenome(SummarizedExperiment::rowRanges(dist_se))
```

---

readColMM *readColMM*

Description

Modified the ‘Matrix::readMM‘ function for reading matrices stored in the Harwell-Boeing or MatrixMarket formats but only reads selected column.

Usage

```
readColMM(file, which.col, chunk = 1000L)
```

Arguments

file	the name of the file to be read from as a character scalar. Those storing matrices in the MatrixMarket format usually end in ".mtx".
which.col	An integer scalar, the column index
chunk	An integer scalar indicating the chunk size to use, i.e., number of rows to read at any one time.

Details

See readMM

Value

A sparse matrix object that inherits from the "Matrix" class which the original dimensions.To get the vector of the specified column, one need to subset the matrix to select the column with the same index.

Author(s)

Ruqian Lyu

Examples

```
demo_path <-paste0(system.file("extdata",package = "comapr"),"/")
readColMM(file = paste0(demo_path,"s1_chr1_vi.mtx"), which.col=2,chunk=2)
```

`readHapState` *readHapState*

Description

A function that parses the viterbi state matrix (in .mtx format), barcode.txt and snpAnno.txt files for each individual.

Usage

```
readHapState(
  sampleName,
  chroms = c("chr1"),
  path,
  barcodeFile = NULL,
  minSNP = 30,
  minlogllRatio = 200,
  bpDist = 100,
  maxRawCO = 10,
  nmad = 1.5,
  minCellSNP = 200,
  biasTol = 0.45
)
```

Arguments

`sampleName`	the name of the sample to parse which is used as prefix for finding relevant files for the underlying sample
`chroms`	the character vectors of chromosomes to parse. Multiple chromosomes' results will be concated together.
`path`	the path to the files, with name patterns *chrom_vi.mtx, *chrom_viSegInfo.txt, end with slash
`barcodeFile`	if NULL, it is assumed to be in the same directory as the other files and with name sampleName_barcodes.txt
`minSNP`	the crossover(s) will be filtered out if introduced by a segment that has fewer than 'minSNP' SNPs to support.
`minlogllRatio`	the crossover(s) will be filtered out if introduced by a segment that has lower than 'minlogllRatio' to its reversed state.
`bpDist`	the crossover(s) will be filtered out if introduced by a segment that is shorter than 'bpDist' basepairs. It can be a single value or a vector that is the same length and order with 'chroms'.
`maxRawCO`	if a cell has more than 'maxRawCO' number of raw crossovers called across a chromosome, the cell is filtered out
`nmad`	how many mean absolute deviations lower than the median number of SNPs per cellfor a cell to be considered as low coverage cell and filtered Only effective when number of cells are larger than 10. When effective, this or 'minCellSNP', whichever is larger, is applied
`minCellSNP`	the minimum number of SNPs detected for a cell to be kept, used with 'nmads'

biasTol — the SNP's haplotype ratio across all cells is assumed to be 1:1. This argument can be used for removing SNPs that have a biased haplotype. i.e. almost always inferred to be haplotype state 1. It specifies a bias tolerance value, SNPs with haplotype ratios deviating from 0.5 smaller than this value are kept. Only effective when number of cells are larger than 10

Value

a RangedSummarizedExperiment with rowRanges as SNP positions that contribute to crossovers in any cells. colData contains cells annotation including barcodes and sampleName.

Author(s)

Ruqian Lyu

Examples

```
demo_path <- system.file("extdata",package = "comapr")
s1_rse_state <- readHapState(sampleName="s1",chroms=c("chr1"),
path=paste0(demo_path,"/"),
barcodeFile=NULL,minSNP = 0, minlogllRatio = 50,
bpDist = 100,maxRawCO=10,minCellSNP=3)
s1_rse_state
```

---

snp_geno *Markers by genotype results for a group of samples*

Description

Markers by genotype results for a group of samples

Usage

```
data(snp_geno)
```

Format

A data frame with columns:

C57BL.6J genotype of reference mouse train across markers

FVB.NJ..i. genotype of alternative mouse train across markers

POS SNP marker base-pair location

CHR SNP marker chromosome location

X100 a mouse sample

X101 a mouse sample

X102 a mouse sample

X103 a mouse sample

X104 a mouse sample

X105 a mouse sample

X106 a mouse sample

X107 a mouse sample

X108 a mouse sample

X109 a mouse sample

X110 a mouse sample

X111 a mouse sample

X112 a mouse sample

X113 a mouse sample

X92 a mouse sample

X93 a mouse sample

X94 a mouse sample

X95 a mouse sample

X96 a mouse sample

X97 a mouse sample

X98 a mouse sample

X99 a mouse sample

rsID the SNP ID

Source

Statistics Canada. Table 001-0008 - Production and farm value of maple products, annual. http://www5.statcan.gc.ca/cansim/

`snp_geno_gr` *Markers by genotype results for a group of samples*

Description

Markers by genotype results for a group of samples

Usage

```
data(snp_geno_gr)
```

Format

A GRanges object:

X100 a mouse sample

X101 a mouse sample

X102 a mouse sample

X103 a mouse sample

X104 a mouse sample

X105 a mouse sample

X106 a mouse sample

X107 a mouse sample

X108 a mouse sample

X109 a mouse sample

X110 a mouse sample

X111 a mouse sample

X112 a mouse sample

X113 a mouse sample

X92 a mouse sample

X93 a mouse sample

X94 a mouse sample

X95 a mouse sample

X96 a mouse sample

X97 a mouse sample

X98 a mouse sample

X99 a mouse sample

rsID the SNP ID

Source

TBD

`twoSamples` *RangedSummarizedExperiment object containing the Viterbi states SNP markers for samples from two groups. 'colData(twoSamples)' contains the sample group factor.*

Description

RangedSummarizedExperiment object containing the Viterbi states SNP markers for samples from two groups. 'colData(twoSamples)' contains the sample group factor.

Usage

```
data(twoSamples)
```

Format

An object of class `RangedSummarizedExperiment` with 6 rows and 10 columns.

Index

∗ **datasets**
coCount, 8
parents_geno, 24
snp_geno, 33
snp_geno_gr, 34
twoSamples, 35
∗ **internal**
.change_missing, 2
.filterCOsExtra, 3
.label_gt, 4
fill_fail, 14
.change_missing, 2
.filterCOsExtra, 3
.label_gt, 4

bootstrapDist, 5

calGeneticDist, 6
calGeneticDist,GRanges,missing,ANY,ANY,character-method (calGeneticDist), 6
calGeneticDist,GRanges,missing,ANY,ANY,missing-method (calGeneticDist), 6
calGeneticDist,GRanges,numeric,ANY,ANY,character-method (calGeneticDist), 6
calGeneticDist,GRanges,numeric,ANY,ANY,missing-method (calGeneticDist), 6
calGeneticDist,RangedSummarizedExperiment,missing,ANY,character-method (calGeneticDist), 6
calGeneticDist,RangedSummarizedExperiment,missing,ANY,missing-method (calGeneticDist), 6
calGeneticDist,RangedSummarizedExperiment,numeric,ANY,character-method (calGeneticDist), 6
calGeneticDist,RangedSummarizedExperiment,numeric,ANY,missing-method (calGeneticDist), 6
coCount, 8
comapr, 9
combineHapState, 9
correctGT, 10
countBinState, 11
countCOs, 12
countCOs,GRanges-method (countCOs), 12
countCOs,RangedSummarizedExperiment-method (countCOs), 12
countGT, 13

DataTrack, *18*, *20*

fill_fail, 14
filterGT, 14
filterGT,GRanges,numeric,numeric-method (filterGT), 14
filterGT,matrix,numeric,numeric-method (filterGT), 14
findDupSamples, 15

getAFTracks, 16
getCellAFTrack, 17
getCellCORange, 19
getCellDPTrack, 20
getDistortedMarkers, 21
getMeanDPTrack, 22
getSNPDensityTrack, 23

parents_geno, 24
perCellChrQC, 24
perSegDist, 25
perSegChrQC, 26
plotCount, 27
plotCount,GRanges,logical,character-method (plotCount), 27
plotCount,GRanges,logical,missing-method (plotCount), 27
plotCount,GRanges,missing,character-method (plotCount), 27
plotCount,GRanges,missing,missing-method (plotCount), 27
plotCount,RangedSummarizedExperiment,logical,character-method (plotCount), 27
plotCount,RangedSummarizedExperiment,logical,missing-method (plotCount), 27
plotCount,RangedSummarizedExperiment,missing,character-method (plotCount), 27
plotCount,RangedSummarizedExperiment,missing,missing-method (plotCount), 27
plotGeneticDist, 29
plotGTFreq, 30
plotWholeGenome, 30

readColMM, 31

readHapState, 32
readMM, *31*

snp_geno, 33
snp_geno_gr, 34

twoSamples, 35