Package ‘ctDNAtools’

March 4, 2020

Title Analyze Circulating Tumor DNA Sequencing Data

Version 0.4.0

Description Contains tools to analyze minimal residual disease and cell-free DNA fragmentation from aligned sequencing data.
More details on the methods can be found in:
Amjad Alkodsi, Leo Meriranta, Annika Pasanen, Sirpa Leppä (2020) <doi:10.1101/2020.01.27.912790>.

License MIT + file LICENSE

Depends R (>= 3.6.0),

Imports magrittr, dplyr (>= 0.8.3), tidyr (>= 1.0.0), purrr (>= 0.3.2), Rsamtools (>= 2.0.0), assertthat (>= 0.2.1), GenomicRanges, IRanges, GenomeInfoDb, BiocGenerics, BSgenome, GenomicAlignments, rlang (>= 0.4.0), Biostrings, methods, furrr (>= 0.1.0), ellipsis (>= 0.3.0), VariantAnnotation (>= 1.30.1)

Suggests BSgenome.Hsapiens.UCSC.hg19, testthat (>= 2.1.0), ggplot2, knitr, rmarkdown, covr, pkgdown

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

VignetteBuilder knitr

URL `https://github.com/alkodsi/ctDNAtools`

BugReports `https://github.com/alkodsi/ctDNAtools/issues`

NeedsCompilation no

Author Amjad Alkodsi [aut, cre] (<https://orcid.org/0000-0003-3528-4683>)

Maintainer Amjad Alkodsi <`amjad.alkodsi@gmail.com`>

Repository CRAN

Date/Publication 2020-03-04 18:10:02 UTC

R topics documented:

`analyze_fragmentation` *Provides fragment ends analysis*

Description

Calculates the number of fragment ends and the Windowed Protection Score (WPS) in genomic tiles within targets

Usage

```
analyze_fragmentation(bam, targets, tag = "", window_size = 120,
  step_size = 5, min_size = 120, max_size = 180, ...)
```

Arguments

`bam`	the input bam file
`targets`	The targets to restrict the windows within. Must have the columns chr, start and end. In case of whole-genome, specify full chromosomes targets.
`tag`	the RG tag if the bam has more than one sample.
`window_size`	The window (bin) size to use within the targets
`step_size`	The step size to use in case of overlapping bins.
`min_size`	Restrict fragments to this minimum size.
`max_size`	Restrict fragments to this maximum size.
`...`	Other parameters passed to get_fragment_size

Details

Fragment length will extracted from the bam file according to the parameters passed to `get_fragment_size`, and the number of fragment ends, and the Windowed Protection Score (WPS) will be computed in the binned input targets. Binning is done according to the window_size and step_size parameters.

WPS is defined as the number of fragments completely spanning a window (bin) minus the number of fragments with an endpoint within the same window as reported by Snyder et al., Cell 2016.

The output include both the fragment end counts and the WPS in their raw format as well as after adjustment by coverage in the bin.

Minimum and maximum bounds of the fragment size are applied before computing WPS and fragment ends counts.

Value

a data frame with the first three columns having the bins coordinates and other columns having the WPS (raw and adjusted by coverage) and number of fragment ends (raw and adjusted by coverage).

See Also

`get_fragment_size` `bin_fragment_size` `summarize_fragment_size`

Examples

```
data("targets", package = "ctDNAtools")
bamN1 <- system.file("extdata", "N1.bam", package = "ctDNAtools")

## basic usage
analyze_fragmentation(bam = bamN1, targets = targets)

## more options
analyze_fragmentation(
  bam = bamN1, targets = targets,
  step_size = 10, window_size = 50
)
```

`bin_fragment_size` *Gets histogram of fragment lengths from a bam file*

Description

The function first extracts fragment length from a bam file then computes the histogram over defined bins. If normalized is TRUE, the counts per bin will be normalized to the total read counts. Optionally, it can computes the histogram of fragment lengths only for mutated reads (confirmed ctDNA molecules).

Usage

```
bin_fragment_size(bam, mutations = NULL, targets = NULL, tag = "",
  bin_size = 2, custom_bins = NULL, normalized = FALSE,
  min_size = 1, max_size = 400, ...)
```

Arguments

`bam`	path to bam file.
`mutations`	An optional data frame with mutations. Must have the columns CHROM, POS, REF, ALT.
`targets`	a data frame with the target regions to restrict the reads in the bam. Must have three columns: chr, start and end
`tag`	the RG tag if the bam has more than one sample.
`bin_size`	the width of the bin (breaks) of the histogram.
`custom_bins`	A numeric vector for custom breaks to bin the histogram of fragment length. Over-rides bin_size.
`normalized`	A logical, whether to normalize the counts to the total number of reads.
`min_size`	Integer with the lowest fragment length.
`max_size`	Integer with the highest fragment length.
`...`	Other parameters passed to get_fragment_size.

Details

Fragment length will extracted from the bam file according to the parameters passed to `get_fragment_size`, and histogram counts (optionally normalized to total counts) are computed. Both equal histogram bins via bin_size and manually customized bins via custom_bins are supported.

By using an input mutations, the function will bin separately the reads that support variant alleles, reference alleles and other reads.

Value

A data frame with one column for the used breaks and one having the histogram (normalized) counts. If mutations is supplied, the output will have one breaks column and three columns corresponding to variant allele reads, reference allele reads, and other reads. Each row has the count of fragment lengths within the bin and optionally normalized by the total number of reads.

See Also

`get_fragment_size` `analyze_fragmentation` `summarize_fragment_size`

Examples

```
data("targets", package = "ctDNAtools")
data("mutations", package = "ctDNAtools")
bamT1 <- system.file("extdata", "T1.bam", package = "ctDNAtools")
```

```
## basic usage
bin_fragment_size(bam = bamT1)

## with normalization
bin_fragment_size(bam = bamT1, normalized = TRUE)


## binning reads categorized based on mutations ref and alt
bin_fragment_size(bam = bamT1, mutations = mutations)

## Restrict to reads into targets
bin_fragment_size(bam = bamT1, targets = targets)
```

`create_background_panel`

Creates a background panel from a list of bam files

Description

This function scans the targets regions in the list of bam files, and reports the number of reference, non-reference reads for each loci in addition to the non-reference (VAF) allele frequency. Loci with VAF higher than vaf_threshold are masked with NA.

Usage

```
create_background_panel(bam_list, targets, reference,
  vaf_threshold = 0.05, bam_list_tags = rep("", length(bam_list)),
  min_base_quality = 10, max_depth = 1e+05, min_mapq = 20,
  substitution_specific = TRUE)
```

Arguments

`bam_list`	A character vector of paths to bam files.
`targets`	The targets data frame must have the columns chr, start and end.
`reference`	The reference genome as BSgenome object.
`vaf_threshold`	Loci with the fraction of non-reference reads above this value are masked with NA.
`bam_list_tags`	RG tags for the list of bam files. By default, the whole bam file will be used.
`min_base_quality`	The minimum base quality to count a read for a loci.
`max_depth`	Maximum depth for the pileup
`min_mapq`	The minimum mapping quality to count a read for a loci
`substitution_specific`	logical, whether to have the loci by substitutions.

Details

Extracts the depth, variant allele counts and variant allele frequency (VAF) for each genomic position in the input targets across a panel of bam files (e.g. from healthy samples to represent only technical noise). The extracted information can be fed to `create_black_list` in order to extract a black listed loci according to defined criteria

The function support two modes, either loci-specific regardless of the basepair substitution, or substitution-specific where each substitution class (e.g. C>T, C>G) are quantified separately. This behavior is controlled by the substitution_specific parameter.

VAF above vaf_threshold parameters are masked with NA, to exclude real SNPs/mutations.

Since this function can take a long time when the bam_list comprises a large number of bam files, the function supports multi-threading using the furrr and future R packages. All you need to do is call 'plan(multiprocess)' or other multi-threading strategies before calling this function.

Value

A named list having depth, alt and vaf data frames. Each has the same order of loci in rows and the input samples in columns.

See Also

`create_black_list` `test_ctDNA`

Examples

```
## Load example data
data("targets", package = "ctDNAtools")
bamN1 <- system.file("extdata", "N1.bam", package = "ctDNAtools")
bamN2 <- system.file("extdata", "N2.bam", package = "ctDNAtools")
bamN3 <- system.file("extdata", "N3.bam", package = "ctDNAtools")

## Use human reference genome from BSgenome.Hsapiens.UCSC.hg19 library
suppressMessages(library(BSgenome.Hsapiens.UCSC.hg19))

## Use a black list based on loci
bg_panel <- create_background_panel(
  bam_list = c(bamN1, bamN2, bamN3),
  targets = targets, reference = BSgenome.Hsapiens.UCSC.hg19,
  substitution_specific = FALSE
)

bl1 <- create_black_list(bg_panel,
  mean_vaf_quantile = 0.99,
  min_samples_one_read = 2, min_samples_two_reads = 1
)

## Use a substitution-specific black list
bg_panel <- create_background_panel(
  bam_list = c(bamN1, bamN2, bamN3),
  targets = targets, reference = BSgenome.Hsapiens.UCSC.hg19,
  substitution_specific = TRUE
)

bl2 <- create_black_list(bg_panel,
  mean_vaf_quantile = 0.99,
  min_samples_one_read = 2, min_samples_two_reads = 1
)

## Multi-threading (commented)
## library(furrr)
## plan(multiprocess)
## plan(multiprocess, workers = 3)
bg_panel <- create_background_panel(
  bam_list = c(bamN1, bamN2, bamN3),
  targets = targets, reference = BSgenome.Hsapiens.UCSC.hg19,
  substitution_specific = TRUE
)
```

create_black_list — *Creates a black list of genomic loci based on a background panel created from a list of bam files (e.g. healthy samples)*

Description

The function applies criteria on the background panel to extract the noisy genomic loci. Criteria include minimum number of samples having at least one, at least two, or at least n (`n_reads` parameter) non-reference allele. Additionally the quantile of mean VAF above which the loci are considered noisy

Usage

```
create_black_list(background_panel, mean_vaf_quantile = 0.95,
  min_samples_one_read = max(2, ceiling(ncol(background_panel$vaf) *
  0.75)), min_samples_two_reads = max(2,
  ceiling(ncol(background_panel$vaf) * 0.2)), min_samples_n_reads = NA,
  n_reads = NA)
```

Arguments

`background_panel`	A list produced by create_background panel function
`mean_vaf_quantile`	The quantile of mean VAF above which the loci are considered noisy. Use NA to skip this criterion.
`min_samples_one_read`	Loci that at least this number of samples exhibit at least one non-reference reads are considered noisy. Use NA to skip this criterion.

`min_samples_two_reads`	Loci that at least this number of samples exhibit at least two non-reference reads are considered noisy. Use NA to skip this criterion.
`min_samples_n_reads`	Loci that at least this number of samples exhibit at least n non-reference reads (`n_reads` parameter) are considered noisy. Use NA to skip this criterion.
`n_reads`	the number of reads to use in the `min_samples_n_reads` parameter

Value

a character vector of the loci in the black list

See Also

`create_background_panel` `test_ctDNA`

Examples

```
## Load example data
data("targets", package = "ctDNAtools")
bamN1 <- system.file("extdata", "N1.bam", package = "ctDNAtools")
bamN2 <- system.file("extdata", "N2.bam", package = "ctDNAtools")
bamN3 <- system.file("extdata", "N3.bam", package = "ctDNAtools")

## Use human reference genome from BSgenome.Hsapiens.UCSC.hg19 library
suppressMessages(library(BSgenome.Hsapiens.UCSC.hg19))

## Use a black list based on loci
bg_panel <- create_background_panel(
  bam_list = c(bamN1, bamN2, bamN3),
  targets = targets, reference = BSgenome.Hsapiens.UCSC.hg19,
  substitution_specific = FALSE
)

bl1 <- create_black_list(bg_panel,
  mean_vaf_quantile = 0.99,
  min_samples_one_read = 2, min_samples_two_reads = 1,
  min_samples_n_reads = 1, n_reads = 3
)

## Use a substitution-specific black list
bg_panel <- create_background_panel(
  bam_list = c(bamN1, bamN2, bamN3),
  targets = targets, reference = BSgenome.Hsapiens.UCSC.hg19,
  substitution_specific = TRUE
)

bl2 <- create_black_list(bg_panel,
  mean_vaf_quantile = 0.99,
  min_samples_one_read = 2, min_samples_two_reads = 1,
  min_samples_n_read = NA
)
```

`extract_trinucleotide_context`
Extracts the trinucleotide context for a set of mutations

Description

Extracts the trinucleotide context for a set of mutations

Usage

```
extract_trinucleotide_context(mutations, reference, destrand = TRUE)
```

Arguments

`mutations`	A data frame having the mutations. Should have the columns CHROM, POS, REF, ALT.
`reference`	the reference genome in BSgenome format
`destrand`	logical, whether to destrand mutations

Value

A data frame with two columns having the substitutions and the trinucleotide context

Examples

```
data("mutations", package = "ctDNAtools")
## Use human reference genome from BSgenome.Hsapiens.UCSC.hg19 library
suppressMessages(library(BSgenome.Hsapiens.UCSC.hg19))

## with destranding
extract_trinucleotide_context(mutations, BSgenome.Hsapiens.UCSC.hg19)

## without destranding
extract_trinucleotide_context(mutations, BSgenome.Hsapiens.UCSC.hg19,
  destrand = FALSE
)
```

filter_mutations *Filters a set of mutations*

Description

This function Filters a set of mutations given the input black list or the prevalence of their mismatches in a set of bam files. Mutations that have more than min_alt_reads in more than min_samples will be removed when no black list is given.

Usage

```
filter_mutations(mutations, bams = NULL, black_list = NULL,
  tags = rep("", length(bams)), min_alt_reads = 2, min_samples = 2,
  min_base_quality = 20, max_depth = 1e+05, min_mapq = 30,
  substitution_specific = TRUE)
```

Arguments

`mutations`	A data frame with the reporter mutations. Should have the columns CHROM, POS, REF, ALT.
`bams`	a vector of paths to bam files
`black_list`	a character vector of genomic loci of format chr_pos to filter. If not given, the bams will be scanned for mismatches in the mutations loci and the specified thresholds will be applied for filtering.
`tags`	a vector of the RG tags if the bam has more than one sample
`min_alt_reads`	the threshold of read counts showing alternative allele for a sample to be counted
`min_samples`	the threshold of number of samples above which the mutations is filtered
`min_base_quality`	minimum base quality for a read to be counted
`max_depth`	maximum depth above which sampling will happen
`min_mapq`	the minimum mapping quality for a read to be counted
`substitution_specific`	logical, whether to have the loci of black_list by substitutions.

Details

Filter a set of mutations using one of two options:

1. By providing a black list (recommended), which includes a vector of genomic loci chr_pos when substitution_specific is false, or chr_pos_ref_alt when substitutions_specific is true. In this mode, all mutations reported in the black list are simply removed.
2. By providing a set of bam files. The function will run a similar functionality to `create_background_panel` and filter mutations based on the min_alt_reads and min_samples criteria.

This function is called internally in `test_ctDNA` so you likely won't need to use it yourself.

Value

a named list contains:

- ref: vector of read counts of the reference alleles
- alt: vector of read counts of the alternative allele

See Also

`create_black_list` `test_ctDNA` `create_background_panel`

Examples

```
data("mutations", package = "ctDNAtools")
filter_mutations(mutations, black_list = "chr14_106327474_C_G")
```

`get_background_rate` *Computes the background mismatch rate from a bam file*

Description

Runs through the target regions base by base counting the mismatches. Then it divides sum(mismatches)/sum(depths) for all bases in the targets

Usage

```
get_background_rate(bam, targets, reference, vaf_threshold = 0.1,
  tag = "", black_list = NULL, substitution_specific = TRUE,
  min_base_quality = 20, max_depth = 1e+05, min_mapq = 30)
```

Arguments

`bam`	path to bam file
`targets`	a data frame with the target regions. Must have three columns: chr, start and end
`reference`	the reference genome in BSgenome format
`vaf_threshold`	the bases with higher than this VAF threshold will be ignored in the calculation (real mutations)
`tag`	the RG tag if the bam has more than one sample
`black_list`	a character vector of genomic loci of format chr_pos if substitution_specific is false, or chr_pos_ref_alt if substitution_specific is true. The background will be computed on the target regions after excluding black_list loci.
`substitution_specific`	logical, whether to have the loci of black_list by substitutions.
`min_base_quality`	minimum base quality for a read to be counted
`max_depth`	maximum depth above which sampling will happen
`min_mapq`	the minimum mapping quality for a read to be counted

Details

Computes the background rate of the input bam file for all bases in the specified targets. Substitutions-specific rates are also calculated.

Genomic positions having non-reference allele frequency higher than vaf_threshold will be excluded (to exclude SNPs and real mutations).

If a black_list is specified, the positions in the black_list (whether substitution_specific or not) will be excluded before computing the background rate.

Value

a list containing the general mismatch rate and substitution-specific rates

See Also

`create_black_list` `test_ctDNA` `create_background_panel`

Examples

```
## Load example data
data("targets", package = "ctDNAtools")
bamT1 <- system.file("extdata", "T1.bam", package = "ctDNAtools")
bamN1 <- system.file("extdata", "N1.bam", package = "ctDNAtools")
bamN2 <- system.file("extdata", "N2.bam", package = "ctDNAtools")
bamN3 <- system.file("extdata", "N3.bam", package = "ctDNAtools")

## Use human reference genome from BSgenome.Hsapiens.UCSC.hg19 library
suppressMessages(library(BSgenome.Hsapiens.UCSC.hg19))

## basic usage
get_background_rate(bamT1, targets, BSgenome.Hsapiens.UCSC.hg19)

## more options
get_background_rate(bamT1, targets, BSgenome.Hsapiens.UCSC.hg19,
  min_base_quality = 30, min_mapq = 40, vaf_threshold = 0.05
)

## with blacklist
bg_panel <- create_background_panel(
  bam_list = c(bamN1, bamN2, bamN3),
  targets = targets, reference = BSgenome.Hsapiens.UCSC.hg19,
  substitution_specific = TRUE
)

bl2 <- create_black_list(bg_panel,
  mean_vaf_quantile = 0.99,
  min_samples_one_read = 2, min_samples_two_reads = 1
)

get_background_rate(bamT1, targets, BSgenome.Hsapiens.UCSC.hg19,
  black_list = bl2
)
```

get_fragment_size *Gets fragment lengths from a bam file*

Description

A function to extract fragment lengths from a bam file. Optionally, given a mutation data frame, it can categorize read lengths in mutated vs non-mutated reads.

Usage

```
get_fragment_size(bam, mutations = NULL, targets = NULL, tag = "",
  isProperPair = NA, mapqFilter = 30, min_size = 1, max_size = 400,
  ignore_trimmed = TRUE, different_strands = TRUE,
  simple_cigar = FALSE)
```

Arguments

`bam`	path to bam file.
`mutations`	An optional data frame with mutations. Must have the columns CHROM, POS, REF, ALT.
`targets`	a data frame with the target regions to restrict the reads in the bam. Must have three columns: chr, start and end
`tag`	the RG tag if the bam has more than one sample.
`isProperPair`	a logical whether to return only proper pairs (true), only improper pairs (false), or it does not matter (NA).
`mapqFilter`	mapping quality threshold for considering reads.
`min_size`	Integer with the lowest fragment length.
`max_size`	Integer with the highest fragment length.
`ignore_trimmed`	logical, whether to remove reads that have been hard trimmed.
`different_strands`	logical, whether to keep only reads whose mates map to different strand.
`simple_cigar`	logical, whether to include only reads with simple cigar.

Details

Extracts the fragment size of reads in the input bam that satisfy the following conditions:

- Paired, and optionally properly paired depending on the isProperPair parameter.
- Both the reads and mate are mapped.
- Not secondary or supplementary alignment

- Not duplicate
- Passing quality control
- Read and mate on different strands (optional depending on the different_strands parameter)
- Not trimmed (optional depending on the ignore_trimmed parameter), i.e. will keep only reads with the max length.
- Having a simple cigar (optional depending on the simple_cigar parameter).
- Satisfy the mapping quality threshold specified in the mapqFilter parameter.
- Reads and mates on the same chromosome when min_size > 0.

When the input mutations is given, the output will label the reads that support the variant alleles of the mutation in a separate column.

Value

A data frame with the columns:

- Sample: The SM tag in bam or file name
- ID: the read ID
- chr: chromosome
- start: the left most end of either the read or mate
- end: the right most end of either the read or mate.
- size: the fragment size
- category (only if mutations is provided): either ref, alt, or other

See Also

`summarize_fragment_size` `bin_fragment_size` `analyze_fragmentation` `get_mutations_fragment_size`

Examples

```
data("mutations", package = "ctDNAtools")
data("targets", package = "ctDNAtools")
bamT1 <- system.file("extdata", "T1.bam", package = "ctDNAtools")

## basic usage
fs <- get_fragment_size(bam = bamT1)

## More options
fs1 <- get_fragment_size(
  bam = bamT1, isProperPair = TRUE, min_size = 70,
  max_size = 200, ignore_trimmed = FALSE, different_strands = FALSE,
  simple_cigar = TRUE
)

## with mutations input
fs2 <- get_fragment_size(bam = bamT1, mutations = mutations)
```

```
## using targets
fs3 <- get_fragment_size(bam = bamT1, targets = targets)
```

get_mutations_fragment_size

Gets reads fragment lengths for a list of mutations

Description

The function extracts the fragment lengths for the reads holding alternative allele for each mutation in the mutations data frame.

Usage

```
get_mutations_fragment_size(bam, mutations, tag = "",
  min_base_quality = 20, min_mapq = 30, ...)
```

Arguments

`bam`	path to bam file.
`mutations`	Data frame with mutations. Must have the columns CHROM, POS, REF, ALT.
`tag`	the RG tag if the bam has more than one sample.
`min_base_quality`	minimum base quality when extracting reads covering mutations.
`min_mapq`	minimum mapping quality when extracting reads covering mutations.
`...`	Other parameters passed to get_fragment_size.

Details

Fragment length will extracted from the bam file according to the parameters passed to `get_fragment_size`, and the fragment size of the reads that map to the ref and alt alleles of each mutation in the input will be returned.

Value

A list with length equal to the number of mutations. Each element contains a list with two elements ref and alt each having an integer vector of fragment lengths

See Also

`get_fragment_size`

Examples

```
data("mutations", package = "ctDNAtools")
bamT1 <- system.file("extdata", "T1.bam", package = "ctDNAtools")

mfs <- get_mutations_fragment_size(bam = bamT1, mutations = mutations[1:2, ])
```

---

get_mutations_read_counts

Counts ref and alt reads for a set of mutations

Description

Counts ref and alt reads for a set of mutations in a bam file

Usage

```
get_mutations_read_counts(mutations, bam, tag = "",
  min_base_quality = 20, max_depth = 1e+05, min_mapq = 30)
```

Arguments

`mutations`	A data frame with the reporter mutations. Should have the columns CHROM, POS, REF, ALT.
`bam`	path to bam file
`tag`	the RG tag if the bam has more than one sample
`min_base_quality`	minimum base quality for a read to be counted
`max_depth`	maximum depth above which sampling will happen
`min_mapq`	the minimum mapping quality for a read to be counted

Details

Quantifies the reference and variant alleles for the input mutations in the input bam file. Useful for forced calling mutations.

Value

a named list contains: ref, vector of read counts of the reference alleles, and alt, vector of read counts of the alternative allele

See Also

`get_mutations_read_names` `test_ctDNA` `get_mutations_fragment_size`

Examples

```
data("mutations", package = "ctDNAtools")
bamT1 <- system.file("extdata", "T1.bam", package = "ctDNAtools")
get_mutations_read_counts(mutations = mutations[1:3, ], bam = bamT1)
```

---

get_mutations_read_names

Gets names of the reads showing reference and alternative allele of a list of mutations

---

Description

This function extracts the names of the reads in a bam file that support the variant and reference alleles of the input mutations

Usage

```
get_mutations_read_names(bam, mutations, min_base_quality = 20,
  tag = "", min_mapq = 30)
```

Arguments

`bam`	path to bam file
`mutations`	A data frame containing the mutations. Must have the columns CHROM, POS, REF, ALT.
`min_base_quality`	integer specifying the minimum base quality for reads to be included.
`tag`	the RG tag if the bam has more than one sample
`min_mapq`	integer specifying the minimum mapping quality for reads to be included

Details

Returns the IDs of the read that cover the input mutations (ref and alt alleles).

Value

A list with length equal to the number of mutations. Each element is a character vector with the read names.

See Also

`get_mutations_read_counts` `get_mutations_fragment_size` `test_ctDNA`

Examples

```
data("mutations", package = "ctDNAtools")
bamT1 <- system.file("extdata", "T1.bam", package = "ctDNAtools")
get_mutations_read_names(bam = bamT1, mutations = mutations[1:3, ])
```

merge_mutations_in_phase

Collapses mutations in phase into one event

Description

Given a mutations data frame and a bam file, this function collapses mutations in phase identified by the ID_column into one event. While doing that, it ignores the reads that support both the reference and alternative alleles for different mutations in phase.

Usage

```
merge_mutations_in_phase(mutations, bam, tag = "",
  ID_column = "phasingID", min_base_quality = 20, min_mapq = 30)
```

Arguments

`mutations`	A data frame with the reporter mutations. Should have the columns CHROM, POS, REF, ALT.
`bam`	path to bam file
`tag`	the RG tag if the bam has more than one sample
`ID_column`	The name of the column in mutations data.frame that has the IDs for mutations in phase. NA values will be filled automatically by unique mutation identifiers.
`min_base_quality`	minimum base quality for a read to be counted
`min_mapq`	integer specifying the minimum mapping quality for reads to be included.

Details

Mutations in phase are those that are supported by the same reads (same allele). The function doesn't identify mutations in phase, but rather use an ID column in the input whose name is specified by ID_column to tell which mutations are in phase.

Since two or more mutations can be supported by the same evidence, this function merges these mutations into one event. The function will also remove the mismatches that are not exhibited in all the covered phased mutations (since this function is developed for the intent of minimal residual disease testing).

The output will include the merged mutations, the probability of purification, which is defined as the number of reads covering at least two mutations in phase divided by the number of informative reads. Informative reads count is the total number of unique reads mapping to the mutations input (including both mutations in phase and other mutations).

Value

A list with the following slots:

out: A data frame that has the columns:

- Phasing_id: the ID of the mutations/event.
- ref: number of reference reads.
- alt: number of alternative reads.
- n_reads_multi_mutation: Number of reads that span more than one mutation in phase.
- all_reads: total number of reads.
- multi_support: number of reads that support the alt allele of multiple mutations in phase.

purification_prob: Probability of purification: sum(n_reads_multi_mutation)/sum(all_reads)

multi_support: Number of multi-support reads in all mutations/events

informative_reads: Number of unique reads covering the mutations/events

See Also

`test_ctDNA` `get_mutations_read_names`

Examples

```
data("mutations", package = "ctDNAtools")
bamT1 <- system.file("extdata", "T1.bam", package = "ctDNAtools")
merge_mutations_in_phase(mutations = mutations[5:10, ], bam = bamT1, ID_column = "PHASING")
```

`mutations` *Example mutations data to use with ctDNAtools package*

Description

Includes 10 mutations in chr14 immunoglobulin region.

Usage

```
mutations
```

Format

A data frame with 10 rows and 5 columns:

CHROM	chromosome
POS	genomic position
REF	Reference allele
ALT	Alternative allele
PHASING	Common ID for mutations in phase

Source

In-house

summarize_fragment_size — *Summarizes fragment size in defined genomic regions*

Description

Summarizes fragment size in defined genomic regions

Usage

```
summarize_fragment_size(bam, regions, tag = "",
  summary_functions = list(Mean = mean, Median = median), ...)
```

Arguments

`bam`	the input bam file
`regions`	data frame containing the genomic regions. Must have the columns chr, start and end.
`tag`	the RG tag if the bam has more than one sample.
`summary_functions`	a named list containing the R functions used for summarization, e.g. mean, sd.
`...`	Other parameters passed to `get_fragment_size`

Details

Fragment size for reads that are paired (optionally properly paired), whose both mates are mapped, not secondary or supplementary alignment, not duplicates, passed quality control, and satisfy mapq threshold will be used for summarization. The reads that overlap the specified regions will be summarized by the specified summary_functions. Overlaps consider fragments to span the left most to the right most coordinate from either the read or the mate. Minimum and maximum bounds of the fragment size will be applied before summarization.

Value

a data frame with the first column having the regions in the format of chr:start-end, and other columns correspond to summary_functions.

See Also

`get_fragment_size` `bin_fragment_size` `analyze_fragmentation`

Examples

```
data("targets", package = "ctDNAtools")
bamT1 <- system.file("extdata", "T1.bam", package = "ctDNAtools")

## binning the target in arbitrary way
## Note that regions don't need to be bins,
## they can be any regions in the genome
regions <- data.frame(
  chr = targets$chr,
  start = seq(from = targets$start - 200, to = targets$end + 200, by = 30),
  stringsAsFactors = FALSE
)
regions$end <- regions$start + 50

## basic usage
sfs <- summarize_fragment_size(bam = bamT1, regions = regions)

## different summary functions
sfs <- summarize_fragment_size(
  bam = bamT1, regions = regions,
  summary_functions = list(
    Var = var, SD = sd,
    meanSD = function(x) mean(x) / sd(x)
  )
)
```

`targets` *Example Genomic targets to use with ctDNAtools package*

Description

Includes 1 target region in chr14.

Usage

```
targets
```

Format

A data frame with 1 row and 3 columns:

chr chromosome

start start genomic position

end end genomic position

Source

In-house

`test_ctDNA` *Tests the ctDNA positivity of a sample*

Description

Given a set of reporter mutation, this functions counts the reads matching the reporter mutations in the sample to be tested, estimates the mismatch rate for the sample to be tested, and then runs a Monte Carlo simulation test to determine whether the tested sample is positive or negative.

Usage

```
test_ctDNA(mutations, bam, targets, reference, tag = "",
  ID_column = NULL, black_list = NULL, substitution_specific = TRUE,
  vaf_threshold = 0.1, min_base_quality = 30, max_depth = 1e+05,
  min_mapq = 40, bam_list = NULL, bam_list_tags = rep("",
  length(bam_list)), min_alt_reads = 1,
  min_samples = ceiling(length(bam_list)/10), n_simulations = 10000,
  pvalue_threshold = 0.05, seed = 123,
  informative_reads_threshold = 10000)
```

Arguments

`mutations`	A data frame with the reporter mutations. Should have the columns CHROM, POS, REF, ALT.
`bam`	path to bam file
`targets`	a data frame with the target regions. Must have three columns: chr, start and end
`reference`	the reference genome in BSgenome format
`tag`	the RG tag if the bam has more than one sample
`ID_column`	The name of the column that contains the ID of mutations in phase. All mutations in Phase should have the same ID in that column.
`black_list`	a character vector of genomic loci to filter. The format is chr_pos if substitution_specific is false or chr_pos_ref_alt if substitution_specific is true. If given, will override `bam_list`.
`substitution_specific`	logical, whether to have the loci of black_list by substitutions.
`vaf_threshold`	When calculating the background rate, the bases with higher than this VAF threshold will be ignored (real mutations/SNPs).
`min_base_quality`	minimum base quality for a read to be counted
`max_depth`	maximum depth above which sampling will happen
`min_mapq`	the minimum mapping quality for a read to be counted
`bam_list`	A vector containing the paths to bam files used to filter mutations. Mutations that have more than min_alt_reads in more than min_samples will be filtered. Using black_list is more recommended.

`bam_list_tags`	the RG tags for the bams included in bams list.
`min_alt_reads`	When bam_list is provided, this sets the minimum number of alternative allele reads for a sample to be counted.
`min_samples`	When number of samples having more than `min_alt_reads` exceeds this number, the mutation will be filtered.
`n_simulations`	the number of Monte Carlo simulations.
`pvalue_threshold`	the p-value threshold used to decide positivity or negativity.
`seed`	the random seed to make the Monte Carlo simulations reproducible.
`informative_reads_threshold`	the number of informative reads (unique reads mapping to specified mutations) under which the test will be undetermined.

Details

This is the main function to test minimal residual disease by ctDNA positivity in a follow-up sample (e.g. after treatment). The inputs include a bam file for the follow-up sample to be tested, a list of reporter mutations (detected for example before treatment in a ctDNA positive sample), and an optional black_list (recommended to use) computed from a list of bam files of healthy-like samples or bam_list of the bam_files to use instead of black_list.

The workflow includes the following steps:

1. Filtering mutations (optional but recommended): The mutations in the input will be filtered removing the ones reported in the black list. If bam_list is provided, the mutations will be filtered according to the min_samples and min_alt_reads parameters. See `filter_mutations`.

2. The background rate will be computed for the input bam. The black list will be plugged in to exclude the black-listed loci when computing the background rate. The black list can be substitution_specific or not, but in both cases, the background rate will be adjusted accordingly. See `get_background_rate`.

3. Counting reference and alternative alleles of the reporter mutations in the bam file.

4. Merging mutations in phase (optional but recommended): If the ID_column is specified in the mutations input, mutations with the same ID (in phase) will be merged. While doing so, it is expected that real traces of mutations will be exhibited jointly in the reads spanning phased mutations, whereas artifacts will not show in all the covered mutations in phase. Therefore, the mismatches that map only to a subset of the mutations in phase but not the others (which are covered and show reference allele) will be removed. This will lead to reduction of the observed mismatches, and therefore, the computed background rate will be adjusted accordingly: new rate = old rate * (1 - purification_probability). See `merge_mutations_in_phase`.

5. Monte Carlo test: this approach was used by Newman et al., Nature Biotechnology 2016.

- Given N reporter mutations each with depth D_i, randomly sample variant allele reads X_i under the background rate p using binomial distribution X_i ~ Binom(D_i, p).
- Repeat `n_simuations` times.
- Count the number of simulations, where simulated data equal or exceed observed data in jointly two measurements: (1) the average VAF for the N mutations, and (2) the number of mutations with non-zero VAF.

    - Compute an empirical p-value as (#successes + 1)/(#simulations + 1)

6. Make a decision: If number of informative reads is lower than `informative_reads_threshold` parameter, the decision will be undetermined. Otherwise, the `pvalue_threshold` parameters will be used to determine positivity or negativity.

Value

a data frame with the following columns:

- sample: The sample name taken from SM field in the bam file or file base name
- n_mutations: The number of mutations used in the test.
- n_nonzero_alt: Number of mutations that have at least one read supporting alternative allele.
- total_alt_reads: Total number of reads supporting alternative alleles of all mutations in input.
- mutations_filtered: The number of filtered mutations.
- background_rate: The background rate of the tested sample (after all adjustments)
- informative_reads: The number of unique reads covering the mutations used.
- multi_support_reads: The number of reads that support more than one mutations in phase. Non-zero values is a sign of positivity not used in the p-value calculation.
- pvalue: The empirical p-value from the Monte Carlo test.
- decision: The decision can be positive, negative or undetermined.

See Also

`get_background_rate` `merge_mutations_in_phase` `create_black_list` `create_background_panel` `filter_mutations`

Examples

```
## Load example data
data("mutations", package = "ctDNAtools")
data("targets", package = "ctDNAtools")
bamT1 <- system.file("extdata", "T1.bam", package = "ctDNAtools")
bamT2 <- system.file("extdata", "T2.bam", package = "ctDNAtools")
bamN1 <- system.file("extdata", "N1.bam", package = "ctDNAtools")
bamN2 <- system.file("extdata", "N2.bam", package = "ctDNAtools")
bamN3 <- system.file("extdata", "N3.bam", package = "ctDNAtools")

## Use human reference genome from BSgenome.Hsapiens.UCSC.hg19 library
suppressMessages(library(BSgenome.Hsapiens.UCSC.hg19))

## basic usage
test_ctDNA(
  mutations = mutations, bam = bamT1,
  targets = targets, reference = BSgenome.Hsapiens.UCSC.hg19,
  n_simulation = 100
)

## More options
```

```
test_ctDNA(
  mutations = mutations, bam = bamT1,
  targets = targets, reference = BSgenome.Hsapiens.UCSC.hg19,
  n_simulation = 100, min_base_quality = 20, min_mapq = 30,
  vaf_threshold = 0.05
)

## Use phasing information
test_ctDNA(
  mutations = mutations, bam = bamT2,
  targets = targets, reference = BSgenome.Hsapiens.UCSC.hg19,
  n_simulation = 100, ID_column = "PHASING"
)

## Use a black list based on loci
bg_panel <- create_background_panel(
  bam_list = c(bamN1, bamN2, bamN3),
  targets = targets, reference = BSgenome.Hsapiens.UCSC.hg19,
  substitution_specific = FALSE
)

bl1 <- create_black_list(bg_panel,
  mean_vaf_quantile = 0.99,
  min_samples_one_read = 2, min_samples_two_reads = 1
)

test_ctDNA(
  mutations = mutations, bam = bamT1,
  targets = targets, reference = BSgenome.Hsapiens.UCSC.hg19,
  n_simulation = 100, ID_column = "PHASING", black_list = bl1,
  substitution_specific = FALSE
)

## Use a substitution-specific black list
bg_panel <- create_background_panel(
  bam_list = c(bamN1, bamN2, bamN3),
  targets = targets, reference = BSgenome.Hsapiens.UCSC.hg19,
  substitution_specific = TRUE
)

bl2 <- create_black_list(bg_panel,
  mean_vaf_quantile = 0.99,
  min_samples_one_read = 2, min_samples_two_reads = 1
)

test_ctDNA(
  mutations = mutations, bam = bamT1,
  targets = targets, reference = BSgenome.Hsapiens.UCSC.hg19,
  n_simulation = 100, ID_column = "PHASING", black_list = bl2,
  substitution_specific = TRUE
)
```

vcf_to_mutations_df *Helper function to read a vcf into the required format of mutations data frame*

Description

Uses VariantAnnotation::readVcfAsVRanges to read the vcf file, which return variants in a format that each row is one variant. If the vcf has multiple samples, the samples will be appended by rows. Provide a sample_name to return only the variants belonging to the sample of interest. Once you use this function, make sure that all the variants are relevant. The function will only return SNVs.

Usage

```
vcf_to_mutations_df(vcf, sample_name = NULL, ...)
```

Arguments

`vcf`	the path to vcf file
`sample_name`	a character(1) when provided, return only variants from this sample
`...`	other options passed to VariantAnnotation::readVcfAsVRanges

Examples

```
vcf <- system.file("extdata", "chr22.vcf.gz", package="VariantAnnotation")
vcf_to_mutations_df(vcf, sample_name = "HG00096")
```

Index