Package: fRagmentomics (via r-universe)

October 14, 2026

Title Extract Fragmentomics Features and Mutational Status

Version 1.1.0

Description A user-friendly R package that enables the characterization of each cfDNA fragment overlapping one or multiple mutations of interest, starting from a sequencing file containing aligned reads (BAM file). fRagmentomics supports multiple mutation input formats (e.g., VCF, TSV, or string ``chr:pos:ref:alt'' representation), accommodates one-based and zero-based genomic conventions, handles mutation representation ambiguities, and accepts any reference file and species in FASTA format. For each cfDNA fragment, fRagmentomics outputs its size, its 3' and 5' sequences, and its mutational status. Optionally, when users set apply_bcftools_norm = TRUE, fRagmentomics invokes the external command-line tool bcftools norm to left-align and normalize variants. If bcftools is not found on the system PATH while this option is enabled, the function errors. The package does not install external software; see the INSTALL file for per-OS instructions.

URL https://github.com/ElsaB-Lab/fRagmentomics

BugReports https://github.com/ElsaB-Lab/fRagmentomics/issues

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

biocViews Software, Genetics, VariantDetection, IndelDetection, Sequencing, DNASeq, Alignment, MultipleSequenceAlignment

Suggests ragg, covr, testthat (>= 3.0.0), knitr, rmarkdown (>= 1.14), BiocStyle

VignetteBuilder knitr

Config/testthat/edition 3

Imports Biostrings, data.table, dplyr, future, future.apply, GenomeInfoDb, GenomicRanges, ggh4x, ggplot2, ggseqlogo, IRanges, purrr, RColorBrewer, readr, rlang, Rsamtools (>= 2.4.0), S4Vectors, VariantAnnotation, scales, stringr, tibble, tidyr

SystemRequirements (optional) bcftools (>= 1.21) for VCF left-alignment/normalization via 'bcftools norm'

Config/pak/sysreqs make libbz2-dev libicu-dev liblzma-dev libpng-dev libxml2-dev libssl-dev libx11-dev xz-utils zlib1g-dev

Repository https://bioc.r-universe.dev

Date/Publication 2026-04-28 13:06:32 UTC

RemoteUrl https://github.com/bioc/fRagmentomics

RemoteRef HEAD

RemoteSha 69857f385283d4d00554d0824eecea2cf6cb5659

Contents

plot_freq_barplot 2
plot_ggseqlogo_meme 5
plot_motif_barplot 8
plot_size_distribution 11
run_fRagmentomics 13

Index **17**

`plot_freq_barplot` *Plot Overall Nucleotide Frequency*

Description

Creates a bar plot to compare the overall proportion of each nucleotide (A/C/G/T; optional 'Other') in the end motifs. Error bars show 95% confidence intervals.

Usage

```
plot_freq_barplot(
  df_fragments,
  end_motif_5p = "Fragment_Bases_5p",
  end_motif_3p = "Fragment_Bases_3p",
  motif_type = "Both",
  motif_size = 3,
  col_z = "Fragment_Status_Simple",
  vals_z = NULL,
  ...,
  colors_z = "Dark2",
```

```
  title = NULL,
  output_path = NA_character_,
 ggsave_params = list(width = 14, height = 5, units = "in", dpi = 300, bg = "white"),
  show_pvalue = FALSE,
  drop_non_acgt = TRUE
)
```

Arguments

df_fragments	The input data frame containing fragment sequence data.
end_motif_5p	Character string. Column name for 5' end sequences.
end_motif_3p	Character string. Column name for 3' end sequences.
motif_type	Character string. Which ends to analyze: 'Start', 'End', or 'Both'.
motif_size	A single integer (>= 1) specifying the k-mer length to analyze.
col_z	Character string or 'NULL'. Column name for grouping. If 'NULL', all fragments are pooled.
vals_z	A character vector of group names from 'col_z' to include. If 'NULL', all unique groups in 'col_z' are used.
...	Additional aesthetics/arguments passed to ggplot2::geom_col() and ggplot2::geom_errorbar(). (e.g., 'alpha', 'position', or 'width').
colors_z	A character vector of colors for the groups, or a single RColorBrewer palette name (e.g., 'Set2'). Named vectors are aligned to 'vals_z'.
title	Character or 'NA'. Plot title. If 'NULL', 'NA', or empty, a default title is used.
output_path	Character or 'NA'. If provided and non-empty, the plot is saved to this path.
ggsave_params	A named list of arguments passed to ggplot2::ggsave().
show_pvalue	Logical. If 'TRUE' and there are at least two groups, append a global Chi-squared p-value to the caption.
drop_non_acgt	Logical. If 'FALSE', characters other than A/C/G/T are tallied into an 'Other' category.

Value

A 'ggplot' object. If 'output_path' is provided and non-empty, the plot is saved to file and the function returns 'invisible(NULL)'.

Examples

```
## --- Create a dataset for demonstration ---
set.seed(42)

# Helper to generate random DNA sequences with base bias
generate_biased_dna <- function(n_seq, len, prob) {
  bases <- c("A", "C", "G", "T")
  replicate(n_seq, paste(sample(bases, len, replace = TRUE, prob = prob), collapse = ""))
}
```

```
# 50 'MUT' fragments biased toward 'C'
df_mut <- data.frame(
  Fragment_Bases_5p = generate_biased_dna(50, 10, prob = c(0.2, 0.5, 0.15, 0.15)),
  Fragment_Bases_3p = generate_biased_dna(50, 10, prob = c(0.2, 0.5, 0.15, 0.15)),
  Fragment_Status_Simple = "MUT"
)

# 50 'WT' fragments biased toward 'G'
df_wt <- data.frame(
  Fragment_Bases_5p = generate_biased_dna(50, 10, prob = c(0.15, 0.15, 0.5, 0.2)),
  Fragment_Bases_3p = generate_biased_dna(50, 10, prob = c(0.15, 0.15, 0.5, 0.2)),
  Fragment_Status_Simple = "WT"
)

# Combine into a single data frame
example_df <- rbind(df_mut, df_wt)

## --- Function calls ---

# 1) Default plot: compare MUT vs WT for 3-mers from both ends
p1 <- plot_freq_barplot(example_df)
p1

# 2) First-nucleotide only (k = 1) on 5' end, with custom colors
p2 <- plot_freq_barplot(
  df_fragments = example_df,
  motif_type   = "Start",
  motif_size   = 1,
  colors_z     = c("MUT" = "#d95f02", "WT" = "#1b9e77"),
  title        = "5' First Base Composition"
)
p2

# 3) Ungrouped: overall nucleotide frequencies across all fragments
p3 <- plot_freq_barplot(example_df, col_z = NULL, title = "Overall Composition")
p3

# 4) Subset of groups (if you had >2 groups, e.g., 'MUT', 'WT', 'AMB')
p4 <- plot_freq_barplot(
  df_fragments = example_df,
  vals_z       = c("MUT", "WT")
)
p4

# 5) Include non-ACGT characters tallied as 'Other'
example_df$Fragment_Bases_5p[1:3] <- c("NNNNNNNNNN", "ACGTNACGTN", "TTTNAAAAAA")
p5 <- plot_freq_barplot(example_df,
  motif_size = 2, drop_non_acgt = FALSE,
  title = "Including 'Other' (non-ACGT)"
)
p5

# 6) Save to file with specific dimensions
```

```
# plot_freq_barplot(
#   df_fragments = example_df,
#   output_path  = file.path(tempdir(), 'nucleotide_frequency.png'),
#   ggsave_params = list(width = 7, height = 5, units = 'in', dpi = 300, bg = 'white')
# )
```

`plot_ggseqlogo_meme` *Plot sequence motif composition*

Description

Creates a sequence logo plot showing the proportion of each nucleotide at each position, with flexible grouping/faceting.

Usage

```
plot_ggseqlogo_meme(
  df_fragments,
  end_motif_5p = "Fragment_Bases_5p",
  end_motif_3p = "Fragment_Bases_3p",
  motif_type = "Both",
  motif_size = 3,
  col_z = "Fragment_Status_Simple",
  vals_z = NULL,
  colors_z = NULL,
  title = NULL,
  output_path = NA_character_,
 ggsave_params = list(width = 12, height = 6, units = "in", dpi = 300, bg = "white"),
  ...
)
```

Arguments

`df_fragments`	Data frame containing fragment sequence data.
`end_motif_5p`	Character. Column name for 5' end sequences.
`end_motif_3p`	Character. Column name for 3' end sequences.
`motif_type`	Character. One of 'Start', 'End', or 'Both'.
`motif_size`	Integer (>=1). Length of the motif to analyze.
`col_z`	Character or NULL. Grouping/faceting column. If NULL, all fragments are pooled.
`vals_z`	Character vector or NULL. Subset of groups from 'col_z' to include. If NULL, all unique groups are used.
`colors_z`	NULL (use ggseqlogo defaults), a single RColorBrewer palette name (e.g., 'Dark2'), or a named vector for 'A/C/G/T', e.g. 'c(A='#1B9E77', C='#D95F02', G='#7570B3', T='#E7298A')'.

title	Character or NA. Plot title; if NULL/NA/'NA'/empty, a default title is used.
output_path	Character or NA. If provided and non-empty, the plot is saved to this file.
ggsave_params	Named list passed to ggplot2::ggsave().
...	Extra arguments forwarded to ggseqlogo::ggseqlogo() (e.g., 'stack_width', 'font', or 'col_scheme').

Value

A 'ggplot' object (invisibly NULL if saved).

Examples

```
## --- Create a dataset for demonstration ---
set.seed(42)

# Helper to generate random DNA sequences with base bias
generate_biased_dna <- function(n_seq, len, prob) {
  bases <- c("A", "C", "G", "T")
  replicate(n_seq, paste(sample(bases, len, replace = TRUE, prob = prob), collapse = ""))
}

# 50 'MUT' fragments biased toward 'C' at the ends
df_mut <- data.frame(
  Fragment_Bases_5p = generate_biased_dna(50, 10, prob = c(0.2, 0.5, 0.15, 0.15)),
  Fragment_Bases_3p = generate_biased_dna(50, 10, prob = c(0.2, 0.5, 0.15, 0.15)),
  Fragment_Status_Simple = "MUT"
)

# 50 'WT' fragments biased toward 'G' at the ends
df_wt <- data.frame(
  Fragment_Bases_5p = generate_biased_dna(50, 10, prob = c(0.15, 0.15, 0.5, 0.2)),
  Fragment_Bases_3p = generate_biased_dna(50, 10, prob = c(0.15, 0.15, 0.5, 0.2)),
  Fragment_Status_Simple = "WT"
)

# Combine into a single data frame
example_df <- rbind(df_mut, df_wt)

## --- Function calls ---

# 1) Default plot: 3-mer from both 5' and 3' ends, separated by a dash,
#    faceted by group ('MUT' and 'WT').
p1 <- plot_ggseqlogo_meme(example_df)
p1

# 2) Single-end motif: 5-mer from the 5' end only.
p2 <- plot_ggseqlogo_meme(
  df_fragments = example_df,
  motif_type   = "Start",
  motif_size   = 5,
  title        = "5' motif (k=5)"
```

```
)
p2

# 3) Custom colors using an RColorBrewer palette (first 4 colors mapped to A/C/G/T).
#    Note: the '-' separator in 'Both' is not colored.
p3 <- plot_ggseqlogo_meme(
  df_fragments = example_df,
  motif_type   = "Both",
  motif_size   = 3,
  colors_z     = "Dark2",
  title        = "Both ends (palette = Dark2)"
)
p3

# 4) Fully custom nucleotide colors (named vector).
custom_cols <- c(A = "#1B9E77", C = "#D95F02", G = "#7570B3", T = "#E7298A")
p4 <- plot_ggseqlogo_meme(
  df_fragments = example_df,
  motif_type   = "Start",
  motif_size   = 3,
  colors_z     = custom_cols,
  title        = "Custom nucleotide colors"
)
p4

# 5) Ungrouped: analyze all fragments together (single facet).
p5 <- plot_ggseqlogo_meme(example_df, col_z = NULL, title = "All fragments pooled")
p5

# 6) Passing extra ggseqlogo options via '...' (e.g., stack width and font)
p6 <- plot_ggseqlogo_meme(
  df_fragments = example_df,
  motif_type   = "End",
  motif_size   = 4,
  stack_width  = 0.9,
  font         = "helvetica_regular",
  title        = "3' motif (k=4, custom stack width)"
)
p6

# 7) Save to file (commented out for CRAN)
# out_file <- file.path(tempdir(), 'motif_logo.png')
# plot_ggseqlogo_meme(
#   df_fragments  = example_df,
#   motif_type    = 'Both',
#   motif_size    = 3,
#   title         = 'Saved motif logo',
#   output_path   = out_file,
#   ggsave_params = list(width = 7, height = 5, units = 'in', dpi = 300, bg = 'white')
# )
```

`plot_motif_barplot` *Plot 3-base motif proportions with various representations*

Description

Creates a bar plot to visualize the proportion of 3-base motifs at fragment ends. Supports grouped analysis and three different visual representations: hierarchical faceting by base, log2 fold change, or side-by-side motifs.

Usage

```
plot_motif_barplot(
  df_fragments,
  end_motif_5p = "Fragment_Bases_5p",
  end_motif_3p = "Fragment_Bases_3p",
  motif_type = "Both",
  motif_start = NULL,
  col_z = "Fragment_Status_Simple",
  vals_z = NULL,
  representation = "split_by_base",
  ...,
  colors_z = NULL,
  title = NULL,
  output_path = NA_character_,
 ggsave_params = list(width = 10, height = 6, units = "in", dpi = 300, bg = "white")
)
```

Arguments

`df_fragments`	The input dataframe containing fragment sequence data.
`end_motif_5p`	Character string. Column name for 5' end sequences.
`end_motif_3p`	Character string. Column name for 3' end sequences.
`motif_type`	Character string. Which ends to analyze: 'Start', 'End', or 'Both'.
`motif_start`	Optional character vector ('A','C','G','T') to filter motifs by their starting base.
`col_z`	Character string. Column name for grouping. If NULL, no grouping is applied.
`vals_z`	A character vector of group names from 'col_z' to include. If NULL, all unique groups in 'col_z' are used.
`representation`	Character string. The type of plot to generate. • 'split_by_base' (default): A proportion plot with hierarchical axes, splitting motifs by each base position into facets. • 'differential': A log2 fold change plot comparing two groups. • 'split_by_motif': A proportion plot with motifs on the x-axis, with bars for different groups placed side-by-side.
`...`	Additional arguments passed on to 'ggplot2::geom_bar()'.

colors_z	Colors for the representation: • For 'split_by_base': 4 colors for A/C/G/T, or a single RColorBrewer palette name. • For 'differential': 2 colors for 'Positive'/'Negative' (named vector or palette). • For 'split_by_motif': colors per group (palette, unnamed vector, or a vector named by group names).
title	Character or NA. Plot title; if NULL/NA/'NA'/empty, a default title is used.
output_path	Character or NA. If provided and non-empty, the plot is saved to this file.
ggsave_params	A named list of arguments passed to 'ggplot2::ggsave()'.

Value

A ggplot object.

Examples

```
## --- Create a dataset for demonstration ---
set.seed(42)

# Helper function to generate random DNA sequences with a bias
generate_biased_dna <- function(n_seq, len, prob) {
  bases <- c("A", "C", "G", "T")
  replicate(n_seq, paste(sample(bases, len, replace = TRUE, prob = prob), collapse = ""))
}

# Create 50 'MUT' fragments with a high proportion of motifs starting with 'C'
df_mut <- data.frame(
  Fragment_Bases_5p = generate_biased_dna(50, 10, prob = c(0.2, 0.5, 0.15, 0.15)),
  Fragment_Bases_3p = generate_biased_dna(50, 10, prob = c(0.2, 0.5, 0.15, 0.15)),
  Fragment_Status_Simple = "MUT"
)

# Create 50 'WT' fragments with a high proportion of motifs starting with 'G'
df_wt <- data.frame(
  Fragment_Bases_5p = generate_biased_dna(50, 10, prob = c(0.15, 0.15, 0.5, 0.2)),
  Fragment_Bases_3p = generate_biased_dna(50, 10, prob = c(0.15, 0.15, 0.5, 0.2)),
  Fragment_Status_Simple = "WT"
)

# Combine into a single dataframe
example_df <- rbind(df_mut, df_wt)

## --- Function Calls for Each Representation ---

# 1. Hierarchical Plot (representation = 'split_by_base')
# This is the default. It creates nested facets for each base position.
p1 <- plot_motif_barplot(
  df_fragments   = example_df,
  representation = "split_by_base"
)
```

```
p1

# You can also filter this plot to show only motifs starting with certain bases.
p1_filtered <- plot_motif_barplot(
  df_fragments   = example_df,
  representation = "split_by_base",
  motif_start    = c("C", "G"),
  title          = "Motifs starting with C/G"
)
p1_filtered

# Optional: customize colors for the 2nd base (A/C/G/T) in split_by_base
p1_colors <- plot_motif_barplot(
  df_fragments   = example_df,
  representation = "split_by_base",
  colors_z       = c(A = "#FD96A9", C = "#E88B00", G = "#0D539E", T = "#6CAE75")
)
p1_colors

# 2. Differential Plot (representation = 'differential')
# Shows log2 fold-change in motif proportions between two groups (needs exactly two groups).
p2 <- plot_motif_barplot(
  df_fragments   = example_df,
  representation = "differential",
  vals_z         = c("MUT", "WT"),
  colors_z       = c(Positive = "#66C2A5", Negative = "#E78AC3"),
  title          = "MUT vs WT (log2FC)"
)
p2

# 3. Side-by-side Motif Plot (representation = 'split_by_motif')
# Motifs on the x-axis; bars for each group shown side-by-side.
p3 <- plot_motif_barplot(
  df_fragments   = example_df,
  representation = "split_by_motif",
  colors_z       = "Set2" # or a vector named by group names
)
p3

# 4. Save the default hierarchical plot (commented for CRAN)
# out_file1 <- file.path(tempdir(), 'motif_split_by_base.png')
# plot_motif_barplot(
#   df_fragments   = example_df,
#   representation = 'split_by_base',
#   title          = 'Motif proportions (hierarchical)',
#   output_path    = out_file1,
#   ggsave_params  = list(width = 8, height = 6, units = 'in', dpi = 300, bg = 'white')
# )

# 5. Save the differential plot with custom dimensions (commented for CRAN)
# out_file2 <- file.path(tempdir(), 'motif_differential.png')
# plot_motif_barplot(
#   df_fragments   = example_df,
```

```
#   representation = 'differential',
#   vals_z         = c('MUT', 'WT'),
#   title          = 'Differential motif usage',
#   output_path    = out_file2,
#   ggsave_params  = list(width = 12, height = 8, units = 'in', dpi = 300, bg = 'white')
# )
```

`plot_size_distribution` *Plot Fragment Size Distribution*

Description

Generates a plot visualizing the distribution of fragment lengths. Supports grouping by a categorical variable and can represent the distribution as a histogram, a density plot, or an overlay of both. The legend displays the sample size (N) per group. Groups with fewer than 2 observations are automatically removed when `show_density = TRUE`.

Usage

```
plot_size_distribution(
  df_fragments,
  size_col = "Fragment_Size",
  col_z = "Fragment_Status_Simple",
  vals_z = NULL,
  histo_args = list(),
  density_args = list(),
  colors_z = NULL,
  show_histogram = FALSE,
  show_density = TRUE,
  x_limits = c(0, 600),
  histogram_binwidth = 5,
  show_nuc_peaks = TRUE,
  title = NULL,
  output_path = NA_character_,
 ggsave_params = list(width = 10, height = 7, units = "in", dpi = 300, bg = "white")
)
```

Arguments

`df_fragments`	The input dataframe containing the fragment data.
`size_col`	A character string specifying the name of the numeric column that contains the fragment lengths.
`col_z`	A character string specifying the name of the column to use for grouping the data. If NULL, no grouping is applied.
`vals_z`	An optional character vector to filter and display only specific groups from `col_z`. If NULL, all groups are used.

histo_args	A named list of additional arguments passed to ggplot2::geom_histogram().
density_args	A named list of additional arguments passed to ggplot2::geom_density().
colors_z	A character vector of colors for the groups (named or unnamed), or a single string naming an RColorBrewer palette.
show_histogram	Logical. If TRUE, a histogram layer is added.
show_density	Logical. If TRUE, a density plot layer is added.
x_limits	Optional numeric vector of length 2 to set the x-axis limits (e.g., c(0, 700)).
histogram_binwidth	Numeric value specifying the bin width for the histogram.
show_nuc_peaks	Logical. If TRUE, adds dashed vertical lines for nucleosome peaks (mono/di/tri).
title	Character or NA. Plot title; if NULL/NA/'NA'/empty, a default title is used.
output_path	Character or NA. If provided and non-empty, the plot is saved to this file.
ggsave_params	A named list of arguments passed to ggplot2::ggsave().

Value

A ggplot object representing the size distribution plot (invisibly NULL if saved).

Examples

```
## --- Create a dataset for demonstration ---
set.seed(42)

# Generate fragment sizes for two groups with different distributions
# 'MUT' group: N=100, shorter fragments
mut_sizes <- rnorm(100, mean = 150, sd = 20)

# 'WT' group: N=150, centered around the mononucleosome peak
wt_sizes <- rnorm(150, mean = 170, sd = 25)

# Add some larger, dinucleosomal fragments to both groups
di_nuc_sizes <- rnorm(30, mean = 330, sd = 30)

# Combine into a single dataframe
example_df_size <- data.frame(
  Fragment_Size = c(mut_sizes, wt_sizes, di_nuc_sizes),
  Fragment_Status_Simple = c(
    rep("MUT", 100),
    rep("WT", 150),
    sample(c("MUT", "WT"), 30, replace = TRUE)
  )
)
# Ensure all fragment sizes are positive
example_df_size <- example_df_size[example_df_size$Fragment_Size > 0, ]

## --- Plotting Examples ---

# 1) Default plot: grouped density with nucleosome peaks.
```

```
p1 <- plot_size_distribution(example_df_size)
p1

# 2) Histogram only: add transparency so overlapping bars are visible.
p2 <- plot_size_distribution(
  df_fragments   = example_df_size,
  show_histogram = TRUE,
  show_density   = FALSE,
  histo_args     = list(alpha = 0.6)
)
p2

# 3) Combined: overlay density curves and histograms.
p3 <- plot_size_distribution(
  df_fragments   = example_df_size,
  show_histogram = TRUE,
  show_density   = TRUE,
  histo_args     = list(alpha = 0.4)
)
p3

# 4) Ungrouped + zoomed x-axis + no nucleosome peaks.
p4 <- plot_size_distribution(
  df_fragments   = example_df_size,
  col_z          = NULL,
  x_limits       = c(50, 400),
  show_nuc_peaks = FALSE,
  title          = "All fragments (zoomed)"
)
p4

# 5) Custom colors using an RColorBrewer palette.
p5 <- plot_size_distribution(
  df_fragments = example_df_size,
  colors_z     = "Set2",
  title        = "Fragment size (Set2 palette)"
)
p5

# 6) Save to file (commented for CRAN):
# out_png <- file.path(tempdir(), 'size_distribution.png')
# plot_size_distribution(
#   df_fragments  = example_df_size,
#   output_path   = out_png,
#   ggsave_params = list(width = 8, height = 6, units = 'in', dpi = 300, bg = 'white'),
#   title         = 'Size distribution (saved)'
# )
```

---

run_fRagmentomics *Analyze fragments*

Description

This is the main function of the package. It provides an end-to-end pipeline for analyzing the allelic state of individual DNA fragments covering specific genomic variants. It takes a list of mutations and an aligned sequencing file (BAM) as input, processes each fragment in parallel, and returns a detailed data frame of results.

Usage

```
run_fRagmentomics(
  mut,
  bam,
  fasta,
  sample_id = NA_character_,
  neg_offset_mate_search = -600,
  pos_offset_mate_search = 600,
  one_based = TRUE,
 flag_bam_list = list(isPaired = TRUE, isProperPair = NA, isUnmappedQuery = FALSE,
  hasUnmappedMate = FALSE, isMinusStrand = NA, isMateMinusStrand = NA, isFirstMateRead
  = NA, isSecondMateRead = NA, isSecondaryAlignment = FALSE, isSupplementaryAlignment =
    FALSE, isNotPassingQualityControls = NA, isDuplicate = NA),
  report_bam_info = FALSE,
  report_softclip = FALSE,
  report_5p_3p_bases_fragment = 5,
  remove_softclip = FALSE,
  retain_fail_qc = FALSE,
  apply_bcftools_norm = FALSE,
  tmp_folder = tempdir(),
  output_path = NA_character_,
  verbose = FALSE,
  n_cores = 1
)
```

Arguments

mut	Path to a .vcf or .tsv file or string representation chr:pos:ref:alt of a mutation.
bam	Path to a BAM file.
fasta	Path to the FASTA file for the reference sequence used for generating the BAM file.
sample_id	Sample identifier.
neg_offset_mate_search	Integer. Use in read_bam. Represents the number of nucleotides to extend upstream (negative direction) from the position of interest when querying the BAM file with Rsamtools. his extension ensures that paired reads are retrieved, even if only one mate overlaps the queried position.
pos_offset_mate_search	Integer. Use in read_bam.
one_based	Boolean. TRUE if fasta is in one based. False if in 0 based.

`flag_bam_list` A named list of logicals for filtering reads based on their SAM flag NA = Filter is ignored, TRUE = The read MUST have this flag, FALSE = The read MUST NOT have this flag.

`report_bam_info`
Boolean. Whether to include the bam information.

`report_softclip`
Boolean. Whether to include the number of soft-clipped bases at the fragment extremities in the output.

`report_5p_3p_bases_fragment`
Integer. Whether to include N fragment extremity bases in the output.

`remove_softclip`
Boolean. If set to TRUE, trim soft-clipped bases from the 5' end of Read 5p and from the 3' end of Read 3p.

`retain_fail_qc` Boolean. If set to TRUE, retain fragments that failed the various quality checks in the output.

`apply_bcftools_norm`
Boolean. If set to TRUE, apply bcftools norm on each input variant to normalize it. Require that bcftools command is installed and available in the PATH.

`tmp_folder` Character vector for the temporary folder path.

`output_path` Character vector for the fragmentomics table output path.

`verbose` Boolean. If set to TRUE, print all the warnings and the prints.

`n_cores` Number of cores for parallel computation.

Details

The function executes a multi-step workflow for each variant provided in the 'mut' input:

1. **Input Validation**: All parameters are rigorously checked for correctness (e.g., file existence, data types). Required file indices ('.bai', '.fai') are created automatically if missing.
2. **Variant Normalization**: The input variants are parsed and normalized into a canonical, left-aligned representation using a combination of VCF-style indel padding and the external 'bcftools norm' command.
3. **BAM Read Extraction**: For each normalized variant, the function efficiently queries the BAM file to retrieve all read pairs that cover the genomic locus.
4. **Parallel Fragment Processing**: The core analysis is performed in parallel using the 'future' framework. Each unique DNA fragment is processed by the 'extract_fragment_features' worker function to determine its size, quality metrics, and mutation status (e.g., 'MUT', 'WT', 'AMB', 'N/I').
5. **VAF Calculation**: After all fragments for a variant are processed, the Variant Allele Frequency (VAF) is calculated.
6. **Output Generation**: Results from all variants are aggregated into a single data frame. If a value for `output_path` is provided, this data frame is also written to a tab-separated file.

Value

A `DataFrame` (S4Vectors) containing extracted fragment-level information.

Examples

```
# --- 1. Locate Example Files ---
# The package includes small example files to demonstrate its functionality.
# We locate them using system.file().
mut_file <- system.file(
  "extdata/mutation", "cfdna-egfr-del_chr7_55241864_55243064_10k.mutations.tsv",
  package = "fRagmentomics"
)
bam_file <- system.file(
  "extdata/bam", "cfdna-egfr-del_chr7_55241864_55243064_10k.bam",
  package = "fRagmentomics"
)
fasta_file <- system.file(
  "extdata/fasta", "hg19_chr7_55231864_55253064.fa",
  package = "fRagmentomics"
)

# --- 2. Run the Analysis ---
# This single call runs the full analysis pipeline on the example data.
# The output file is written to a temporary location to avoid cluttering
# the working directory. We use n_cores = 1L for examples.
results <- run_fRagmentomics(
  mut = mut_file,
  bam = bam_file,
  fasta = fasta_file,
  sample_id = "cfdna-egfr-del",
  n_cores = 1L
)

# --- 3. View the Results ---
# Print the first few rows of the output data frame to see the results.
head(results)
```

Index

ggplot2::geom_col(), *3*
ggplot2::geom_density(), *12*
ggplot2::geom_errorbar(), *3*
ggplot2::geom_histogram(), *12*
ggplot2::ggsave(), *3*, *6*, *12*
ggseqlogo::ggseqlogo(), *6*

plot_freq_barplot, 2
plot_ggseqlogo_meme, 5
plot_motif_barplot, 8
plot_size_distribution, 11

run_fRagmentomics, 13