Package 'MutationalPatterns'

February 21, 2025

Type Package

Title Comprehensive genome-wide analysis of mutational processes

Description Mutational processes leave characteristic footprints in genomic DNA. This package provides a comprehensive set of flexible functions that allows researchers to easily evaluate and visualize a multitude of mutational patterns in base substitution catalogues of e.g. healthy samples, tumour samples, or DNA-repair deficient cells. The package covers a wide range of patterns including: mutational signatures, transcriptional and replicative strand bias, lesion segregation, genomic distribution and association with genomic features, which are collectively meaningful for studying the activity of mutational processes. The package works with single nucleotide variants (SNVs), insertions and deletions (Indels), double base substitutions (DBSs) and larger multi base substitutions (MBSs). The package provides functionalities for both extracting mutational signatures de novo and determining the contribution of previously identified mutational signatures on a single sample level. MutationalPatterns integrates with common R genomic analysis workflows and allows easy association with (publicly available) annotation data.

Version 3.17.0

Date 2024-04-05

License MIT + file LICENSE

URL https://doi.org/doi:10.1186/s12864-022-08357-3

Imports stats, S4Vectors, BiocGenerics (>= 0.18.0), BSgenome (>= 1.40.0), VariantAnnotation (>= 1.18.1), dplyr (>= 0.8.3), tibble(>= 2.1.3), purrr (>= 0.3.2), tidyr (>= 1.0.0), stringr (>= 1.4.0), magrittr (>= 1.5), ggplot2 (>= 2.1.0), pracma (>= 1.8.8), IRanges (>= 2.6.0), GenomeInfoDb (>= 1.12.0), Biostrings (>= 2.40.0), ggdendro (>= 0.1-20), cowplot (>= 0.9.2), ggalluvial (>= 0.12.2), RColorBrewer, methods

Depends R (>= 4.2.0), GenomicRanges (>= 1.24.0), NMF (>= 0.20.6)

Suggests BSgenome.Hsapiens.UCSC.hg19 (>= 1.4.0), BiocStyle (>= 2.0.3), TxDb.Hsapiens.UCSC.hg19.knownGene (>= 3.2.2), biomaRt (>= 2.28.0), gridExtra (>= 2.2.1), rtracklayer (>= 1.32.2), ccfindR (>= 1.6.0), GenomicFeatures, AnnotationDbi, testthat, knitr, rmarkdown

biocViews Genetics, SomaticMutation

ZipData NA

LazyData false

RoxygenNote 7.1.1

Encoding UTF-8

VignetteBuilder knitr

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

git_branch devel

git_last_commit 1bdb319

git_last_commit_date 2024-10-29

Repository Bioconductor 3.21

Date/Publication 2025-02-20

Author Freek Manders [aut] (ORCID: <https://orcid.org/0000-0001-6197-347X>),
Francis Blokzijl [aut] (ORCID: <https://orcid.org/0000-0002-8084-8444>),
Roel Janssen [aut] (ORCID: <https://orcid.org/0000-0003-4324-5350>),
Jurrian de Kanter [ctb] (ORCID:
<https://orcid.org/0000-0001-5665-3711>),
Rurika Oka [ctb] (ORCID: <https://orcid.org/0000-0003-4107-7250>),
Mark van Roosmalen [cre],
Ruben van Boxtel [aut, cph] (ORCID:
<https://orcid.org/0000-0003-1285-2836>),
Edwin Cuppen [aut] (ORCID: <https://orcid.org/0000-0002-0400-9542>)

Maintainer Mark van Roosmalen <vanBoxtelBioinformatics@prinsesmaximacentrum.nl>

Contents

`binomial_test` *Binomial test for enrichment or depletion testing*

Description

This function performs lower-tail binomial test for depletion and upper-tail test for enrichment

Usage

```
binomial_test(p, n, x, p_cutoffs = 0.05)
```

Arguments

`p`	Probability of success
`n`	Number of trials
`x`	Observed number of successes
`p_cutoffs`	Significance cutoff for the p value. Default: 0.05

Value

A data.frame with direction of effect (enrichment/depletion), P-value and significance asterisks

Examples

```
binomial_test(0.5, 1200, 543)
binomial_test(0.2, 800, 150)
```

`bin_mutation_density` *Bin the genome based on mutation density*

Description

This function splits the genome based on the mutation density. The density is calculated per chromosome. The density is split into bins. The difference in density between subsequent bins is the same for all bins. In other words, the difference in density between bins 1 and 2 is the same as between bins 2 and 3. The function returns a GRangesList. Each GRanges in the list contains the regions associated with that bin. This can be used with the 'split_muts_region()' function.

Usage

```
bin_mutation_density(vcf_list, ref_genome, nrbins = 3, man_dens_cutoffs = NA)
```

Arguments

`vcf_list`	GRangesList or GRanges object.
`ref_genome`	BSgenome reference genome object
`nrbins`	The number of bins in which to separate the genome
`man_dens_cutoffs`	Manual density cutoffs to use.

Value

GRangesList

See Also

Other genomic_regions: `lengthen_mut_matrix()`, `plot_profile_region()`, `plot_spectrum_region()`, `split_muts_region()`

Examples

```
### See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
grl <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

## Load the corresponding reference genome.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library(ref_genome, character.only = TRUE)

## Determine region density
dens_grl <- bin_mutation_density(grl, ref_genome, nrbins = 3)
names(dens_grl) <- c("Low", "Medium", "High")
```

```
## You can also use manual cutoffs. This feature is meant for more
## advanced users. It can be usefull if you want to find highly mutated regions, with
## a consistent cutoff between analyses.
dens_grl_man <- bin_mutation_density(grl, ref_genome, man_dens_cutoffs = c(0, 2e-08, 1))
```

`calculate_lesion_segregation` *Calculate the amount of lesion segregation for a GRangesList or GRanges object.*

Description

This function calculates lesion segregation for a GRangesList or GRanges object. Lesion segregation is a large scale Watson versus Crick strand asymmetry caused by many DNA lesions occurring during a single cell cycle. It was first described in Aitken et al., 2020, Nature. See their paper for a more in-depth discussion of this phenomenon. This function can perform three different types of test to calculate lesion segregation. The first method is unique to this package, while the other two were also used by Aitken et al., 2020. The 'binomial' test is based on how often consecutive mutations are on different strands. The 'wald-wolfowitz' test checks if the strands are randomly distributed. It's not known which method is superior. The 'rl20' test looks at run sizes (The number of consecutive mutations on the same strand). This is less susceptible to local strand asymetries and kataegis, but doesn't generate a p-value.

Usage

```
calculate_lesion_segregation(
  vcf_list,
  sample_names,
  test = c("binomial", "wald-wolfowitz", "rl20"),
  split_by_type = FALSE,
  ref_genome = NA,
  chromosomes = NA
)
```

Arguments

`vcf_list`	GRangesList or GRanges object
`sample_names`	The name of the sample
`test`	The statistical test that should be used. Possible values: * 'binomial' Binomial test based on the number of strand switches. (Default); * 'wald-wolfowitz' Statistical test that checks if the strands are randomly distributed.; * 'rl20' Calculates rl20 value and the genomic span of the associated runs set.;
`split_by_type`	Boolean describing whether the lesion segregation should be calculated for all SNVs together or per 96 substitution context. (Default: FALSE)
`ref_genome`	BSgenome reference genome object. Only needed when split_by_type is TRUE with the binomial test or when using the rl20 test.
`chromosomes`	The chromosomes that are used. Only needed when using the rl20 test.

Details

The amount of lesion segregation is calculated per GRanges object. The results are then combined in a table.

It's possible to calculate the lesion segregation separately per 96 substitution context, when using the binomial test. The results are then automatically added back up together. This can increase sensitivity when a mutational process causes multiple types of base substitutions, which aren't considered to be on the same strand.

When using the rl20 test, this function first calculates the strand runs per chromosome and combines them. It then calculates the smallest set of runs, which together encompass at least 20 percent of the mutations. (This set thus contains the largest runs). The size of the smallest run in this set is the rl20. The genomic span of the runs in this set is also calculated.

Value

A tibble containing the amount of lesions segregation per sample

See Also

plot_lesion_segregation

Other Lesion_segregation: plot_lesion_segregation()

Examples

```
## See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
grl <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

## To reduce the runtime we take only the first two samples
grl <- grl[1:2]
## Set the sample names
sample_names <- c("colon1", "colon2")

## Load the corresponding reference genome.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library(ref_genome, character.only = TRUE)

## Calculate lesion segregation
lesion_segretation <- calculate_lesion_segregation(grl, sample_names)

## Calculate lesion segregation per 96 base type
lesion_segretation_by_type <- calculate_lesion_segregation(grl, sample_names,
  split_by_type = TRUE, ref_genome = ref_genome
)

## Calculate lesion segregation using the wald-wolfowitz test.
lesion_segregation_wald <- calculate_lesion_segregation(grl,
  sample_names,
  test = "wald-wolfowitz"
```

```
)

## Calculate lesion segregation using the rl20.
chromosomes <- paste0("chr", c(1:22, "X"))
lesion_segregation_rl20 <- calculate_lesion_segregation(grl,
  sample_names,
  test = "rl20",
  ref_genome = ref_genome,
  chromosomes = chromosomes
)
```

`cluster_signatures` *Signature clustering function*

Description

Hierarchical clustering of signatures based on cosine similarity

Usage

```
cluster_signatures(signatures, method = "complete")
```

Arguments

`signatures`	Matrix with 96 trinucleotides (rows) and any number of signatures (columns)
`method`	The agglomeration method to be used for hierarchical clustering. This should be one of "ward.D", "ward.D2", "single", "complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC) or "centroid" (= UPGMC). Default = "complete".

Value

hclust object

See Also

`plot_contribution_heatmap`

Examples

```
## Get signatures
signatures <- get_known_signatures()

## See the 'mut_matrix()' example for how we obtained the mutation matrix:
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))
```

```
## Hierarchically cluster the cancer signatures based on cosine similarity
hclust_signatures <- cluster_signatures(signatures)

## Plot dendrogram
plot(hclust_signatures)
```

`context_potential_damage_analysis` *Potential damage analysis for the supplied mutational contexts*

Description

The ratio of possible 'stop gain', 'mismatches', 'synonymous mutations' and 'splice site mutations' is counted per mutational context. This is done for the supplied ENTREZ gene ids. This way it can be determined how damaging a mutational context could be. N gives the total number of possible mutations per context.

Usage

```
context_potential_damage_analysis(
  contexts,
  txdb,
  ref_genome,
  gene_ids,
  verbose = FALSE
)
```

Arguments

`contexts`	Vector of mutational contexts to use for the analysis.
`txdb`	Transcription annotation database
`ref_genome`	BSgenome reference genome object
`gene_ids`	Entrez gene ids
`verbose`	Boolean. Determines whether progress is printed. (Default: FALSE)

Details

The function works by first selecting the longest transcript per gene. The coding sequence (cds) of this transcript is then assembled. Next, the function loops over the reference contexts. For each context (and it's reverse complement), all possible mutation locations are determined. Splice site mutations are removed at this stage. It's also determined whether these locations are the first, second or third base of the cds codon (mut loc). Each unique combination of codon and mut loc is then counted. For each combination the reference amino acid and the possible alternative amino acids are determined. By comparing the reference and alternative amino acids, the number of 'stop_gains', 'mismatches' and 'synonymous mutations' is determined. This is then normalized per mutation context. For example, mutations with the ACA context could be located in the third position of a

codon like TAC. This might happen 200 times in the supplied genes. This TAC codon could then be mutated in either a TAA, TAG or a TAT. The first two of these options would induce a stop codon, while the third one would be synonymous. By summing up all codons the number of stop_gains', 'mismatches' and 'synonymous mutations' is determined per mutation context.

For mismatches the blosum62 score is also calculated. This is a score based on the BLOSUM62 matrix, that describes how similar two amino acids are. This score is normalized over the total amount of possible mismatches. A lower score means that the amino acids in the mismatches are more dissimilar. More dissimilar amino acids are more likely to have a detrimental effect.

To identify splice sites, sequences around the splice locations are used instead of the cds. The 2 bases 5' and 2 bases 3' of a splice site are considered to be splice site mutation locations.

Value

A tibble with the ratio of 'stop gain', 'mismatch', 'synonymous' and 'splice site' mutations per mutation context.

Examples

```
## See the 'mut_matrix()' example for how we obtained the
## mutation matrix information:
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))

contexts <- rownames(mut_mat)

## Load the corresponding reference genome.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library(ref_genome, character.only = TRUE)

## Load the transcription annotation database
## You can obtain the database from the UCSC hg19 dataset using
## Bioconductor:
# BiocManager::install("TxDb.Hsapiens.UCSC.hg19.knownGene")
library("TxDb.Hsapiens.UCSC.hg19.knownGene")
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

## Here we will use the Entrez Gene IDs from several cancer
## genes. In practice you might want to use larger gene lists,
## but here we only use a few to keep the runtime low.
## In this example we are using:
## TP53, KRAS, NRAS, BRAF, BRCA2
gene_ids <- c(7157, 3845, 4893, 673, 675)

## Run the function
context_potential_damage_analysis(contexts, txdb, ref_genome, gene_ids)

## The function can provide updates about its progress.
## This can be usefull when it's running slowly,
## which can happen when you are using many gene_ids.
## To reduce the example runtime, we don't re-run the analysis, but only show the command
```

```
## context_potential_damage_analysis(contexts, txdb, ref_genome, gene_ids, verbose = TRUE)
```

---

convert_sigs_to_ref *Convert tissue specific signature exposures to reference*

Description

This function converts tissue specific signature contributions into reference signature contributions. This works on SNV signatures from SIGNAL. It uses a conversion matrix to do the conversion. The output can include possible artifact signatures.

Usage

```
convert_sigs_to_ref(fit_res)
```

Arguments

fit_res	Named list with signature contributions and reconstructed mutation matrix

Value

The input fit_res, but with converted signature contributions.

Examples

```
## See the 'mut_matrix()' example for how we obtained the mutation matrix:
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))

## Get tissue specific signatures
signatures <- get_known_signatures(source = "SIGNAL", sig_type = "tissue", tissue_type = "Skin")

## Fit tissue specific signatures
fit_res <- fit_to_signatures(mut_mat, signatures)

## Convert the tissue specific signatures exposures to reference
fit_res <- convert_sigs_to_ref(fit_res)
```

cos_sim *Cosine similarity function*

Description

Calculate the cosine similarity between two vectors of the same length. The cosine similarity is a value between 0 (distinct) and 1 (identical) and indicates how much two vectors are alike.

Usage

```
cos_sim(x, y)
```

Arguments

x	Vector 1 of length n
y	Vector 2 of length n

Value

Cosine similarity value; a value between 0 and 1

Examples

```
x <- c(1.1, 2.1, 0.2, 0.1, 2.9)
y <- c(0.9, 1.9, 0.5, 0.4, 3.1)
cos_sim(x, y)
```

cos_sim_matrix *Compute all pairwise cosine similarities between mutational profiles/signatures*

Description

Computes all pairwise cosine similarities between the mutational profiles provided in the two mutation count matrices. The cosine similarity is a value between 0 (distinct) and 1 (identical) and indicates how much two vectors are alike.

Usage

```
cos_sim_matrix(mut_matrix1, mut_matrix2)
```

Arguments

mut_matrix1	mutation count matrix (dimensions: a mutation features X n samples)
mut_matrix2	96 mutation count matrix (dimensions: a mutation features X m samples)

Value

Matrix with pairwise cosine similarities (dimensions: n mutational profiles X m mutational profiles)

See Also

mut_matrix, fit_to_signatures, plot_cosine_heatmap

Examples

```
## Get signatures
signatures <- get_known_signatures()

## See the 'mut_matrix()' example for how we obtained the mutation matrix:
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))


## Calculate the cosine similarity between each COSMIC signature and each 96 mutational profile
cos_sim_matrix(mut_mat, signatures)
```

---

count_dbs_contexts *Count DBS contexts*

Description

Count DBS contexts

Usage

```
count_dbs_contexts(vcf_list)
```

Arguments

vcf_list	GRanges or GRangesList object containing DBS mutations in which the context was added with get_dbs_context.

Details

Counts the number of DBS per COSMIC context from a GRanges or GRangesList object containing DBS variants. This function applies the count_dbs_contexts_gr function to each gr in its input. It then combines the results in a single tibble and returns this.

Value

A tibble containing the number of DBS per COSMIC context per gr.

See Also

get_dbs_context

Other DBS: get_dbs_context(), plot_compare_dbs(), plot_dbs_contexts(), plot_main_dbs_contexts()

Examples

```
## Get a GRangesList or GRanges object with DBS contexts.
## See 'dbs_get_context' for more info on how to do this.
grl_dbs_context <- readRDS(system.file("states/blood_grl_dbs_context.rds",
  package = "MutationalPatterns"
))

# Count the DBS contexts
count_dbs_contexts(grl_dbs_context)
```

---

count_indel_contexts *Count indel contexts*

Description

Count indel contexts

Usage

```
count_indel_contexts(vcf_list)
```

Arguments

vcf_list	GRanges or GRangesList object containing indel mutations in which the context was added with get_indel_context.

Details

Counts the number of indels per COSMIC context from a GRanges or GRangesList object containing indel mutations. This function applies the count_indel_contexts_gr function to each gr in its input. It then combines the results in a single tibble and returns this.

Value

A tibble containing the number of indels per COSMIC context per gr.

See Also

get_indel_context

Other Indels: get_indel_context(), plot_compare_indels(), plot_indel_contexts(), plot_main_indel_contexts(

Examples

```
## Get a GRangesList or GRanges object with indel contexts.
## See 'indel_get_context' for more info on how to do this.
grl_indel_context <- readRDS(system.file("states/blood_grl_indel_context.rds",
  package = "MutationalPatterns"
))

# Count the indel contexts
count_indel_contexts(grl_indel_context)
```

`count_mbs_contexts` *Count MBS variants grouped by length.*

Description

Count MBS variants grouped by length.

Usage

```
count_mbs_contexts(vcf_list)
```

Arguments

`vcf_list`	GRanges or GRangesList object containing mbs variants.

Details

Counts the number of mbs grouped by length from a GRanges or GRangesList object containing mbs variants. This is used, since a COSMIC context has to our knowledge not yet been defined. This function applies the count_mbs_contexts_gr function to each gr in its input. It then combines the results in a single tibble and returns this.

Value

A tibble containing the number of MBS per MBS length per gr.

See Also

Other MBS: `plot_compare_mbs()`, `plot_mbs_contexts()`

Examples

```
## Get a GRangesList or GRanges object with mbs variants.
mbs_grl <- readRDS(system.file("states/blood_grl_mbs.rds",
  package = "MutationalPatterns"
))

# Count the MBSs
count_mbs_contexts(mbs_grl)
```

`determine_regional_similarity` *Determine regional mutation pattern similarity*

Description

Calculate the cosine similarities between the global mutation profile and the mutation profile of smaller genomic windows, using a sliding window approach. Regions with a very different mutation profile can be identified in this way. This function generally requires many mutations (~100,000) to work properly.

Usage

```
determine_regional_similarity(
  vcf,
  ref_genome,
  chromosomes,
  window_size = 100,
  stepsize = 25,
  extension = 1,
  oligo_correction = FALSE,
  exclude_self_mut_mat = TRUE,
  max_window_size_gen = 2e+07,
  verbose = FALSE
)
```

Arguments

`vcf`	GRanges object
`ref_genome`	BSgenome reference genome object
`chromosomes`	Vector of chromosome/contig names of the reference genome to be plotted.
`window_size`	The number of mutations in a window. (Default: 100)
`stepsize`	The number of mutations that a window slides in each step. (Default: 25)
`extension`	The number of bases, that's extracted upstream and downstream of the base substitutions, to create the mutation matrices. (Default: 1).
`oligo_correction`	Boolean describing whether oligonucleotide frequency correction should be applied. (Default: FALSE)
`exclude_self_mut_mat`	Boolean describing whether the mutations in a window should be subtracted from the global mutation matrix. (Default: TRUE)
`max_window_size_gen`	The maximum size of a window before it is removed. (Default: 20,000,000)
`verbose`	Boolean determining the verbosity of the function. (Default: FALSE)

Details

First a global mutation matrix is calculated using all mutations. Next, a sliding window is used. This means that we create a window containing the first x mutations. The cosine similarity, between the mutation profiles of this window and the global mutation matrix, is then calculated. The window then slides y mutations to the right and the cosine similarity is again calculated. This process is repeated until the final mutation on a chromosome is reached. This process is performed separately per chromosome. Windows that span a too large region of the genome are removed, because they are unlikely to contain biologically relevant information.

The number of mutations that the window slides to the right in each step is called the stepsize. The best stepsize depends on the window size. In general, we recommend setting the stepsize between 25 window size.

The analysis can be performed for trinucleotides contexts, for a larger context, or for just the base substitutions. A smaller context might miss detailed differences in mutation profiles, but is also less noisy. We recommend using a smaller extension when analyzing small datasets.

It's possible to correct for the oligonucleotide frequency of the windows. This is done by calculating the cosine similarity of the oligonucleotide frequency between each window and the genome. The cosine similarity of the mutation profiles is then divided by the oligonucleotide similarity. This ensures that regions with an abnormal oligonucleotide frequency don't show up as having a very different profile. The oligonucleotide frequency correction slows down the function, so we advise the user to keep it off for exploratory analyses and to only turn it on to validate interesting results.

By default the mutations in a window are subtracted from the global mutation matrix, before calculating the cosine similarity. This increases sensitivity, but could also decrease specificity. This subtraction can be turned of with the 'exclude_self_mut_mat' argument.

Value

A "region_cossim" object containing both the cosine similarities and the settings used in this analysis.

See Also

plot_regional_similarity

Other regional_similarity: plot_regional_similarity()

Examples

```
## See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
grl <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

## We pool all the variants together, because the function doesn't work well
## with a limited number of mutations. Still, in practice we recommend to use
## more mutations that in this example.
gr = unlist(grl)

## Specifiy the chromosomes of interest.
```

```
chromosomes <- names(genome(gr)[1:3])

 ## Load the corresponding reference genome.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library(ref_genome, character.only = TRUE)

## Determine the regional similarities. Here we use a small window size to make the function work.
## In practice, we recommend a larger window size.
regional_sims = determine_regional_similarity(gr,
  ref_genome,
  chromosomes,
  window_size = 40,
  stepsize = 10,
  max_window_size_gen = 40000000
)

## Here we use an extensiof of 0 to reduce noise.
## We also turned verbosity on, so you can see at what step the function is.
## This can be useful on large datasets.
regional_sims_0_extension = determine_regional_similarity(gr,
  ref_genome,
  chromosomes,
  window_size = 40,
  stepsize = 10,
  extension = 0,
  max_window_size_gen = 40000000,
  verbose = TRUE
)
```

`enrichment_depletion_test` *Test for enrichment or depletion of mutations in genomic regions*

Description

This function aggregates mutations per group (optional) and performs an enrichment depletion test.

Usage

```
enrichment_depletion_test(x, by = NA, p_cutoffs = 0.05, fdr_cutoffs = 0.1)
```

Arguments

`x`	data.frame result from genomic_distribution()
`by`	Optional grouping variable, e.g. tissue type
`p_cutoffs`	Significance cutoff for the p value. Default: 0.05
`fdr_cutoffs`	Significance cutoff for the fdr. Default: 0.1

Value

data.frame with the observed and expected number of mutations per genomic region per group (by) or sample

See Also

genomic_distribution, plot_enrichment_depletion

Examples

```
## See the 'genomic_distribution()' example for how we obtained the
## following data:
distr <- readRDS(system.file("states/distr_data.rds",
  package = "MutationalPatterns"
))

tissue <- c(rep("colon", 3), rep("intestine", 3), rep("liver", 3))

## Perform the enrichment/depletion test by tissue type.
distr_test <- enrichment_depletion_test(distr, by = tissue)

## Or without specifying the 'by' parameter, to pool all samples.
distr_single_sample <- enrichment_depletion_test(distr)

## Use different significance cutoffs for the pvalue and fdr
distr_strict <- enrichment_depletion_test(distr,
  by = tissue,
  p_cutoffs = 0.01, fdr_cutoffs = 0.05
)

## Use multiple (max 3) significance cutoffs.
## This will vary the number of significance stars.
distr_multistars <- enrichment_depletion_test(distr,
  by = tissue,
  p_cutoffs = c(0.05, 0.01, 0.005),
  fdr_cutoffs = c(0.1, 0.05, 0.01)
)
```

---

extract_signatures *Extract mutational signatures from 96 mutation matrix using NMF*

Description

Decomposes trinucleotide count matrix into signatures and contribution of those signatures to the spectra of the samples/vcf files.

Usage

```
extract_signatures(
  mut_matrix,
  rank,
  nrun = 200,
  nmf_type = c("regular", "variational_bayes"),
  single_core = FALSE,
  fudge = NULL,
  seed = 123456
)
```

Arguments

mut_matrix	96 mutation count matrix
rank	Number of signatures to extract
nrun	Number of iterations, default = 200. A lower number will be faster, but result in less accurate results.
nmf_type	Type of NMF to be used. Possible values: * 'regular' * 'variational_bayes' The 'regular' method comes from the NMF package. The 'variational_bayes' method comes from the ccfindR package. This method uses bayesian inference, which makes it easier to determine the mathematically optimal number of signatures.
single_core	Boolean. If TRUE, it forces the NMF algorithm to use only a single core. This can sometimes prevent issues. Doesn't apply to variational-bayes NMF
fudge	Small positive number that is used for the variational_bayes NMF. Setting this to a small value like 0.0001 can prevent errors from occurring, when extracting many signatures at once. In general, we recommend extracting less signatures when errors occur, but this parameter can be used when that is not an option. Default = NULL.
seed	Random seed used for the regular NMF, default = 123456

Value

Named list of mutation matrix, signatures and signature contribution

See Also

mut_matrix

Examples

```
## See the 'mut_matrix()' example for how we obtained the mutation matrix:
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))

## This function is computationally intensive.
# nmf_res <- extract_signatures(mut_mat, rank = 2)
```

```
## It's also possible to use a variational Bayes method.
## It requires the ccfindR package to work.
# nmf_res <- extract_signatures(mut_mat, rank = 2, nmf_type = "variational_bayes")
```

`fit_to_signatures` *Find optimal nonnegative linear combination of mutation signatures to reconstruct the mutation matrix.*

Description

Find the linear combination of mutation signatures that most closely reconstructs the mutation matrix by solving the nonnegative least-squares constraints problem.

Usage

```
fit_to_signatures(mut_matrix, signatures)
```

Arguments

`mut_matrix`	mutation count matrix (dimensions: x mutation types X n samples)
`signatures`	Signature matrix (dimensions: x mutation types X n signatures)

Value

Named list with signature contributions and reconstructed mutation matrix

See Also

`mut_matrix`,`fit_to_signatures_strict`,`fit_to_signatures_bootstrapped`

Examples

```
## See the 'mut_matrix()' example for how we obtained the mutation matrix:
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))

## Get signatures
signatures <- get_known_signatures()

## Perform the fitting
fit_res <- fit_to_signatures(mut_mat, signatures)

## This will also work for indels and dbs.
## An example is given for indels

## Get The indel counts
## See 'count_indel_contexts()' for more info on how to do this.
```

```
indel_counts <- readRDS(system.file("states/blood_indel_counts.rds",
  package = "MutationalPatterns"
))

## Get signatures
signatures <- get_known_signatures("indel")

fit_to_signatures(indel_counts, signatures)
```

`fit_to_signatures_bootstrapped` *Fit mutational signatures to a mutation matrix with bootstrapping*

Description

Bootstrapping the signature refitting shows how stable the refit is, when small changes are made to the mutation matrix. You can be more confident in the refitting results, when the differences in signature contributions are small between bootstrap iterations.

Usage

```
fit_to_signatures_bootstrapped(
  mut_matrix,
  signatures,
  n_boots = 1000,
  max_delta = 0.004,
  method = c("strict", "regular", "regular_10+", "strict_best_subset",
    "strict_backwards"),
  verbose = TRUE
)
```

Arguments

`mut_matrix`	mutation count matrix (dimensions: x mutation types X n samples)
`signatures`	Signature matrix (dimensions: x mutation types X n signatures)
`n_boots`	Number of bootstrap iterations.
`max_delta`	The maximum difference in original vs reconstructed cosine similarity between two iterations. Only used with method strict.
`method`	The refitting method to be used. Possible values: * 'strict' Uses fit_to_signatures_strict with the default (backwards selection) method; * 'regular' Uses fit_to_signatures; * 'regular_10+' Uses fit_to_signatures, but removes signatures with less than 10 variants.; * 'strict_best_subset' Uses fit_to_signatures_strict with the 'best_subset' method; * 'strict_backwards' Uses fit_to_signatures_strict with the backwards selection method. This is the same as the 'strict' method;
`verbose`	Boolean. If TRUE, the function will show how far along it is.

Details

The mutation matrix is resampled 'n_boots' times. Resampling is done per column (sample) with replacement. The row weights are used as probabilities. On each resampled matrix the 'fit_to_signatures()' or 'fit_to_signatures_strict()' function is applied. In the end a matrix is returned with the contributions for each bootstrap iteration. Each row is a single bootstrap iteration from a single sample. The method you choose determines how strict the signature refitting is. The 'regular' and "regular_10+ methods often suffer from a lot of overfitting, however this is less of an issue when you refit on an limited number of signatures. The 'strict' method suffers less from overfitting, but can suffer from more signature misattribution. The best method will depend on your data and research question.

Value

A matrix showing the signature contributions across all the bootstrap iterations.

See Also

mut_matrix, fit_to_signatures_strict, fit_to_signatures_bootstrapped

Examples

```
## See the 'mut_matrix()' example for how we obtained the mutation matrix:
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))

## Get pre-defined signatures
signatures <- get_known_signatures()

## Fit to signatures with bootstrapping
## Here we use a very low "n_boots" to reduce the runtime.
## For real uses, a much higher value is required.
contri_boots <- fit_to_signatures_bootstrapped(mut_mat,
  signatures,
  n_boots = 2,
  max_delta = 0.004
)

## Use the regular refit method
contri_boots <- fit_to_signatures_bootstrapped(mut_mat,
  signatures,
  n_boots = 2,
  method = "regular"
)
```

`fit_to_signatures_strict` *Fit mutational signatures to a mutation matrix with less overfitting*

Description

Refitting signatures with this function suffers less from overfitting. The strictness of the refitting is dependent on 'max_delta'. A downside of this method is that it might increase signature misattribution. Different signatures might be attributed to similar samples. You can use 'fit_to_signatures_bootstrapped()', to see if this is happening. Using less signatures for the refitting will decrease this issue. Fitting less strictly will also decrease this issue.

Usage

```
fit_to_signatures_strict(
  mut_matrix,
  signatures,
  max_delta = 0.004,
  method = c("backwards", "best_subset")
)
```

Arguments

`mut_matrix`	Mutation count matrix (dimensions: x mutation types X n samples)
`signatures`	Signature matrix (dimensions: x mutation types X n signatures)
`max_delta`	The maximum difference in original vs reconstructed cosine similarity between two iterations.
`method`	The method used to select signatures.

Details

Find a linear non-negative combination of mutation signatures that reconstructs the mutation matrix. Signature selection (feature selection) is done to reduce overfitting. This can be done via either a 'backwards' (default) or 'best_subset' method. The 'backwards' method starts by achieving an optimal reconstruction via 'fit_to_signatures'. The signature with the lowest contribution is then removed and refitting is repeated. This is done in an iterative fashion. Each time the cosine similarity between the original and reconstructed profile is calculated. The 'best_subset' method also starts by achieving an optimal reconstruction via 'fit_to_signatures'. Signature refitting is then repeated for each combination of n-1 signatures, where n is the number of signatures in the signature matrix. The cosine similarity between the original and reconstructed profile is calculated for each combination. The combination with the highest cosine similarity is then chosen. This is done in an iterative fashion for n-2, n-3, ect. With both methods, iterations are stopped when the difference between two iterations becomes more than 'max_delta'. The second-last set of signatures is then used for a final refit.

The 'best_subset' method can result in more accurate results than the 'backwards' method, however it becomes very slow when a large amount of signatures are used for refitting. We recommend only

using the 'best_subset' method when fitting a maximum of 10-15 signatures. When using the 'best_subset' method a lower 'max_delta' should be used, as the expected differences in cosine similarity are reduced.

Value

A list containing a fit_res object, similar to 'fit_to_signatures' and a list of ggplot graphs that for each sample shows in what order the signatures were removed and how this affected the cosine similarity.

See Also

mut_matrix, fit_to_signatures, fit_to_signatures_bootstrapped

Examples

```
## See the 'mut_matrix()' example for how we obtained the mutation matrix:
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))

## Get signatures
signatures <- get_known_signatures()

## Fit to signatures strict
strict_refit <- fit_to_signatures_strict(mut_mat, signatures, max_delta = 0.004)

## fit_res similar to 'fit_to_signatures()'
fit_res <- strict_refit$fit_res

## list of ggplots that shows how the cosine similarity was reduced during the iterations
fig_l <- strict_refit$sim_decay_fig


## Fit to signatures with the best_subset method
## This can be more accurate than the standard backwards method,
## but can only be used with a limited amount of signatures.
## Here we use only 5 signatures to reduce the runtime.
## In practice up to 10-15 signatures could be used.
best_subset_refit <- fit_to_signatures_strict(mut_mat,
   signatures[,1:5],
   max_delta = 0.002,
   method = "best_subset"
)
```

---

genomic_distribution *Find overlaps between mutations and a genomic region.*

Description

Function finds the number of mutations that reside in genomic region and takes surveyed area of genome into account.

Usage

```
genomic_distribution(vcf_list, surveyed_list, region_list)
```

Arguments

vcf_list	GRangesList or GRanges object.
surveyed_list	A GRangesList or a list with GRanges of regions of the genome that have been surveyed (e.g. determined using GATK CallableLoci).
region_list	A GRangesList or a list with GRanges objects containing locations of genomic regions.

Value

A data.frame containing the number observed and number of expected mutations in each genomic region.

See Also

read_vcfs_as_granges

Examples

```
## See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
vcfs <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

## Use biomaRt to obtain data.
## We can query the BioMart database, but this may take a long time,
## so we take some shortcuts by loading the results from our
## examples. The corresponding code for downloading the data can be
## found above the command we run.

# mart="ensemble"
# library(biomaRt)

# regulatory <- useEnsembl(biomart="regulation",
#                          dataset="hsapiens_regulatory_feature",
#                          GRCh = 37)
regulatory <- readRDS(system.file("states/regulatory_data.rds",
  package = "MutationalPatterns"
))

## Download the regulatory CTCF binding sites and convert them to
```

```
## a GRanges object.
# CTCF <- getBM(attributes = c('chromosome_name',
#                              'chromosome_start',
#                              'chromosome_end',
#                              'feature_type_name'),
#               filters = "regulatory_feature_type_name",
#               values = "CTCF Binding Site",
#               mart = regulatory)
#
# CTCF_g <- reduce(GRanges(CTCF$chromosome_name,
#                  IRanges(CTCF$chromosome_start,
#                  CTCF$chromosome_end)))

CTCF_g <- readRDS(system.file("states/CTCF_g_data.rds",
  package = "MutationalPatterns"
))

## Download the promoter regions and conver them to a GRanges object.
# promoter = getBM(attributes = c('chromosome_name', 'chromosome_start',
#                                 'chromosome_end', 'feature_type_name'),
#                  filters = "regulatory_feature_type_name",
#                  values = "Promoter",
#                  mart = regulatory)
# promoter_g = reduce(GRanges(promoter$chromosome_name,
#                     IRanges(promoter$chromosome_start,
#                             promoter$chromosome_end)))

promoter_g <- readRDS(system.file("states/promoter_g_data.rds",
  package = "MutationalPatterns"
))

# flanking = getBM(attributes = c('chromosome_name',
#                                 'chromosome_start',
#                                 'chromosome_end',
#                                 'feature_type_name'),
#                  filters = "regulatory_feature_type_name",
#                  values = "Promoter Flanking Region",
#                  mart = regulatory)
# flanking_g = reduce(GRanges(
#                       flanking$chromosome_name,
#                       IRanges(flanking$chromosome_start,
#                       flanking$chromosome_end)))

flanking_g <- readRDS(system.file("states/promoter_flanking_g_data.rds",
  package = "MutationalPatterns"
))

regions <- GRangesList(promoter_g, flanking_g, CTCF_g)

names(regions) <- c("Promoter", "Promoter flanking", "CTCF")

# Use a naming standard consistently.
seqlevelsStyle(regions) <- "UCSC"
```

```
## Get the filename with surveyed/callable regions
surveyed_file <- system.file("extdata/callableloci-sample.bed",
  package = "MutationalPatterns"
)

## Import the file using rtracklayer and use the UCSC naming standard
library(rtracklayer)
surveyed <- import(surveyed_file)
seqlevelsStyle(surveyed) <- "UCSC"

## For this example we use the same surveyed file for each sample.
surveyed_list <- rep(list(surveyed), 9)

## Calculate the number of observed and expected number of mutations in
## each genomic regions for each sample.
distr <- genomic_distribution(vcfs, surveyed_list, regions)
```

`get_dbs_context` *Get DBS context*

Description

Get the DBS COSMIC context on an GRanges/GRangesList object. It applies the get_dbs_context_gr function to each gr in the input, which works by changing the REF and ALT columns of the GRanges into the COSMIC types.

Usage

```
get_dbs_context(vcf_list)
```

Arguments

`vcf_list`	GRanges/GRangesList

Value

A version of the GRanges/GRangesList object, with modified REF and ALT columns.

See Also

`get_mut_type`, `read_vcfs_as_granges`

Other DBS: `count_dbs_contexts()`, `plot_compare_dbs()`, `plot_dbs_contexts()`, `plot_main_dbs_contexts()`

Examples

```
## Get GRangesList with DBS.
## See 'get_mut_type' or 'read_vcfs_as_granges' for more info on how to do this.
dbs_grl <- readRDS(system.file("states/blood_grl_dbs.rds",
  package = "MutationalPatterns"
))

## Set context DBS
get_dbs_context(dbs_grl)
```

`get_indel_context` *Get indel contexts*

Description

Get indel contexts

Usage

```
get_indel_context(vcf_list, ref_genome)
```

Arguments

`vcf_list`	GRanges or GRangesList object containing Indel mutations. The mutations should be called similarly to HaplotypeCaller.
`ref_genome`	BSgenome reference genome object

Details

Determines the COSMIC context from a GRanges or GRangesList object containing Indel mutations. It applies the get_indel_context_gr function to each gr in the input. It searches for repeat units both to the left and right of the indel.

Value

A modified version of the input grl. In each gr two columns have been added. "muttype" showing the main indel type and "muttype_sub" which shows the subtype. The subtype is either the number of repeats or the microhomology length.

See Also

`read_vcfs_as_granges`, `get_mut_type`

Other Indels: `count_indel_contexts()`, `plot_compare_indels()`, `plot_indel_contexts()`, `plot_main_indel_contexts()`

Examples

```
## Get a GRangesList or GRanges object with only indels.
## See 'read_vcfs_as_granges' or 'get_mut_type' for more info on how to do this.
indel_grl <- readRDS(system.file("states/blood_grl_indel.rds",
  package = "MutationalPatterns"
))

## Load the corresponding reference genome.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library(ref_genome, character.only = TRUE)

## Get the indel contexts
get_indel_context(indel_grl, ref_genome)
```

`get_known_signatures` *Get known signatures*

Description

This function loads a signature matrix of pre-defined signatures. It can retrieve signatures for different types of mutations. It can also retrieve signatures from different sources. Additionally, different signature types can be retrieved. (The possible types are: Reference, tissue specific or drug exposure signatures.) For the COSMIC signatures both GRCh37, GRCh38 and mm10 versions of the signatures can be used. Finally, the user can choose whether to include possible artifacts. If no signatures have been defined for a specific combination of options, then an error is given.

Usage

```
get_known_signatures(
  muttype = c("snv", "dbs", "indel", "tsb_snv"),
  source = c("COSMIC", "SIGNAL", "SPARSE", "COSMIC_v3.1", "COSMIC_v3.2"),
  sig_type = c("reference", "exposure", "tissue"),
  incl_poss_artifacts = FALSE,
  tissue_type = c(NA, "Biliary", "Bladder", "Bone", "Breast", "Cervix", "CNS",
   "Colorectal", "Esophagus", "Head", "Kidney", "Liver", "Lung", "Lymphoid", "Myeloid",
    "Ovary", "Pancreas", "Prostate", "Skin", "Stomach", "Thyroid", "Uterus"),
  genome = c("GRCh37", "GRCh38", "mm10")
)
```

Arguments

`muttype`	The type of mutations. Possible values: * 'snv' (default); * 'dbs'; * 'indel'; * 'tsb_snv' transcription strand bias snv;
`source`	The signature source. Possible values: * 'COSMIC' (default. Currently v3.2); * 'COSMIC_v3.1'; * 'COSMIC_v3.2'; * 'SIGNAL'; * 'SPARSE';
`sig_type`	The type of signature. Possible values: * 'reference' (default); * 'exposure'; * 'tissue';

`incl_poss_artifacts`
: Whether to include possible artifacts. (default: TRUE)

`tissue_type`
: The specific tissue to select signatures from. Can only be used when looking at tissue specific signatures. Keep this at NA to see tissue specific signatures for all tissues.

`genome`
: The genome version that is used. This only works for COSMIC signatures. * 'GRCh37' (default); * 'GRCh38'; * 'mm10';

Details

Possible combinations: COSMIC: - all muttypes. (tsb_snv is the same as in version 3.1) - reference - Can include possible artifacts for SNVs - For the SNVs and DBSs both GRCh37 and GRCh38 versions are available

COSMIC_v3.1: - all muttypes. - reference - Can include possible artifacts for SNVs

SIGNAL: - SNV. (+ DBS, when using exposure signatures.) - all signature types - Can include possible artifacts for reference SNVs

SPARSE: - SNV - reference

Artifacts can be included when using reference signatures for SNVs with COSMIC and SIGNAL

The signatures bundled in this package came from several sources. Please cite the associated papers if you use them.

The COSMIC signatures were downloaded from: https://cancer.sanger.ac.uk/signatures Currently, both version 3.2 and 3.1 are available. Paper: Alexandrov, L.B. et al., 2020, Nature

The SIGNAL signatures were downloaded from: https://signal.mutationalsignatures.com/ They were downloaded on: 03 July 2020. Paper: Andrea Degasperi et al., 2020, Nature Cancer Exposure paper: Jill E Kucab et al., 2019, Cell

The SPARSE signatures were downloaded from: https://www.biorxiv.org/content/10.1101/384834v2 They were downloaded on: 03 July 2020. Paper: Daniele Ramazzotti et al., 2019, Bioarchive

Value

A signature matrix

Examples

```
## Get reference snv signature from COSMIC
get_known_signatures()

## Get reference snv signature from COSMIC,
## including potential artifacts.
get_known_signatures(incl_poss_artifacts = TRUE)

## Get a GRCh38 version of the signatures
get_known_signatures(genome = "GRCh38")

## Get DBS signatures
get_known_signatures("dbs")
```

```
## Get indel signatures
get_known_signatures("indel")

## Get transcription strand bias snv signatures
get_known_signatures("tsb_snv")

## Get COSMIC version 3.1 of the signatures
get_known_signatures(source = "COSMIC_v3.1")

## Get reference signatures from SIGNAL
get_known_signatures(source = "SIGNAL")

## Get reference signatures from SIGNAL,
## including potential artifacts
get_known_signatures(source = "SIGNAL", incl_poss_artifacts = TRUE)

## Get exposure signatures from SIGNAL
get_known_signatures(source = "SIGNAL", sig_type = "exposure")

## Get DBS exposure signatures from SIGNAL
get_known_signatures("dbs", source = "SIGNAL", sig_type = "exposure")

## Get all tissue specific signatures from SIGNAL
get_known_signatures(source = "SIGNAL", sig_type = "tissue")

## Get Bladder specific signatures from SIGNAL
get_known_signatures(
  source = "SIGNAL",
  sig_type = "tissue",
  tissue_type = "Bladder"
)

## If you use an incorrect tissue_type an error is given.

## Get sparse signatures
get_known_signatures(source = "SPARSE")
```

`get_mut_type` *Get variants with mut_type from GRanges*

Description

Get the variants of a certain mutation type from a GRanges or GRangesList object. All other variants will be filtered out. It is assumed that DBS/MBSs are called as separate SNVs. They are merged into single variants. The type of variant can be chosen with type.

Usage

```
get_mut_type(
  vcf_list,
```

```
  type = c("snv", "indel", "dbs", "mbs"),
  predefined_dbs_mbs = FALSE
)
```

Arguments

vcf_list	GRanges/GRangesList
type	The type of variant that will be returned.
predefined_dbs_mbs	Boolean. Whether dbs and mbs variants have been predefined in your vcf. This function by default assumes that dbs and mbs variants are present in the vcf as snvs, which are positioned next to each other. If your dbs/mbs variants are called separately you should set this argument to TRUE. (default = FALSE)

Value

GRanges/GRangesList of the desired mutation type.

See Also

read_vcfs_as_granges

Examples

```
## Get a GRanges list object.
## See 'read_vcfs_as_granges' for more info how to do this.
grl <- readRDS(system.file("states/blood_grl.rds",
  package = "MutationalPatterns"
))

## Here we only use two samples to reduce runtime
grl <- grl[1:2]

## Get a specific mutation type.
snv_grl <- get_mut_type(grl, "snv")
indel_grl <- get_mut_type(grl, "indel")
dbs_grl <- get_mut_type(grl, "dbs")
mbs_grl <- get_mut_type(grl, "mbs")
```

---

get_sim_tb *An S4 generic to get the sim_tb from a region_cossim object.*

Description

An S4 generic to get the sim_tb from a region_cossim object.

An S4 method for the get_sim_tb generic

Usage

```
get_sim_tb(x)

## S4 method for signature 'region_cossim'
get_sim_tb(x)
```

Arguments

x	A region_cossim object
region_cossim	A region_cossim object

Value

A tibble containing the calculated similarities of the windows.

A tibble containing the calculated similarities of the windows.

Methods (by class)

- region_cossim: Get the sim_tb from a region_cossim object.

---

lengthen_mut_matrix *Lengthen mutation matrix*

Description

A mutation_matrix calculated on a GRangesList or GR object modified by 'split_muts_region()', will contain a column per combination of sample and genomic region. In essence different regions are treated as different samples. This function will transform the matrix, so that these regions are instead treated as different mutation types. For example, instead of 'C[C>T]G', you might have the feature 'C[C>T]G Promoter'. The number of rows in the matrix will thus be multiplied by the number of regions. After using 'split_muts_region()', use 'mut_matrix()' to get a mut_matrix that can be used for this function. The result can be plotted with plot_profile_region, but could also be used for NMF, refitting ect.

Usage

```
lengthen_mut_matrix(mut_matrix)
```

Arguments

mut_matrix	Mutation matrix

Value

mut_matrix

See Also

Other genomic_regions: bin_mutation_density(), plot_profile_region(), plot_spectrum_region(), split_muts_region()

Examples

```
## See the 'split_muts_region()' and 'mut_matrix()' examples for how we obtained the
## mutation matrix information:
mut_mat_split_region <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))

long_mut_mat <- lengthen_mut_matrix(mut_mat_split_region)


## This also works on indels:
## See the 'split_muts_region()' and 'count_indels_context()' examples for how we
## obtained the indel counts:
indel_counts_split <- readRDS(system.file("states/blood_indels_counts_split_region.rds",
  package = "MutationalPatterns"
))


## Lengthen the matrix
lengthen_mut_matrix(indel_counts_split)
```

---

merge_signatures *Merge signatures based on cosine similarity*

Description

This function merges signatures based on their cosine similarity. It iteratively merges the two signatures with the highest cosine similarity. Merging is stopped when the maximum cosine similarity is lower than the limit.

Usage

```
merge_signatures(
  signatures,
  cos_sim_cutoff = 0.8,
  merge_char = ";",
  verbose = TRUE
)
```

Arguments

signatures	Signature matrix (dimensions: x mutation types X n signatures)

cos_sim_cutoff	Cutoff for cosine similarity. Signatures are merged when their cosine similarity is higher than the limit. Default: 0.8
merge_char	Character used to merge the signature names. This character shouldn't be in the signature names beforehand. Default: ";"
verbose	Verbosity. If TRUE it shows which signatures got merged. Default: TRUE

Value

Signature matrix (dimensions: x mutation types X n signatures)

Examples

```
## Get signatures
signatures <- get_known_signatures()

## Merge signatures
merge_signatures(signatures)


## Merge signatures using a stricter cutoff
merge_signatures(signatures, cos_sim_cutoff = 0.9)

## Merge signatures using a different merging character
merge_signatures(signatures, merge_char = "_")

## Merge signatures silently
merge_signatures(signatures, verbose = FALSE)
```

MutationalPatterns *MutationalPatterns: an integrative R package for studying patterns in base substitution catalogues*

Description

This package provides an extensive toolset for the characterization and visualization of a wide range of mutational patterns from base substitution catalogues. These patterns include: mutational signatures, transcriptional strand bias, genomic distribution and association with genomic features.

Details

The package provides functionalities for both extracting mutational signatures de novo and inferring the contribution of previously identified mutational signatures. Furthermore, MutationalPatterns allows for easy exploration and visualization of other types of patterns such as transcriptional strand asymmetry, genomic distribution and associations with (publicly available) annotations such as chromatin organization. In addition to identification of active mutation-inducing processes, this approach also allows for determining the involvement of specific DNA repair pathways. For example, presence of a transcriptional strand bias in genic regions may indicate activity of transcription coupled repair.

Author(s)

Francis Blokzijl, Roel Janssen, Ruben van Boxtel, Edwin Cuppen Maintainers: Francis Blokzijl, UMC Utrecht <f.blokzijl@umcutrecht.nl> Roel Janssen, UMC Utrecht <R.R.E.Janssen-10@umcutrecht.nl>

References

Alexandrov,L.B. et al. (2013) Signatures of mutational processes in human cancer. Nature, 500, 415–21.

Blokzijl,F. et al. (2016) Tissue-specific mutation accumulation in human adult stem cells during life. Nature, in press.

Borchers,H.W. (2016) pracma: Practical Numerical Math Functions.

Durinck,S. et al. (2005) BioMart and Bioconductor: A powerful link between biological databases and microarray data analysis. Bioinformatics, 21, 3439–3440.

Gaujoux,R. and Seoighe,C. (2010) A flexible R package for nonnegative matrix factorization. BMC Bioinformatics, 11, 367.

Haradhvala,N.J. et al. (2016) Mutational Strand Asymmetries in Cancer Genomes Reveal Mechanisms of DNA Damage and Repair. Cell, 1–12.

Helleday,T. et al. (2014) Mechanisms underlying mutational signatures in human cancers. Nat. Rev. Genet., 15, 585–598.

Lawrence,M. et al. (2013) Software for computing and annotating genomic ranges. PLoS Comput. Biol., 9, e1003118.

Pleasance,E.D. et al. (2010) A comprehensive catalogue of somatic mutations from a human cancer genome. Nature, 463, 191–196.

See Also

https://github.com/CuppenResearch/MutationalPatterns

`MutationalPatterns-defunct`

Defunct functions in package 'MutationalPattern'

Description

These functions are defunct and no longer available.

Details

Defunct functions are: `mutation_context`, `mutation_types`, `strand_from_vcf`, `explained_by_signatures`

mutations_from_vcf *Retrieve base substitutions from vcf*

Description

A function to extract base substitutions of each position in vcf

Usage

```
mutations_from_vcf(vcf)
```

Arguments

vcf	A CollapsedVCF object

Value

Character vector with base substitutions

See Also

read_vcfs_as_granges

Examples

```
## See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
vcfs <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

muts <- mutations_from_vcf(vcfs[[1]])
```

mut_192_occurrences *Count 192 trinucleotide mutation occurrences*

Description

@details This function is called by mut_matrix_stranded. The 192 trinucleotide context is the 96 trinucleotide context combined with the strands. This function calculates the 192 trinucleotide context for all variants. and then splits these per GRanges (samples). It then calculates how often each 192 trinucleotide context occurs.

Usage

```
mut_192_occurrences(type_context, strand, gr_sizes)
```

Arguments

type_context	result from type_context function
strand	factor with strand information for each position, for example "U" for untranscribed, "T" for transcribed strand, and "-" for unknown
gr_sizes	A vector indicating the number of variants per GRanges

Value

Mutation matrix with 192 mutation occurrences and 96 trinucleotides for two strands

---

mut_96_occurrences *Count 96 trinucleotide mutation occurrences*

Description

@details This function is called by mut_matrix. It calculates the 96 trinucleotide context for all variants and then splits these per GRanges (samples). It then calculates how often each 96 trinucleotide context occurs.

Usage

```
mut_96_occurrences(type_context, gr_sizes)
```

Arguments

type_context	result from type_context function
gr_sizes	A vector indicating the number of variants per GRanges

Value

Mutation matrix with 96 trinucleotide mutation occurrences

---

mut_context *Retrieve context of base substitutions*

Description

A function to extract the bases 3' upstream and 5' downstream of the base substitutions from the reference genome. The user an choose how many bases are extracted.

Usage

```
mut_context(vcf, ref_genome, extension = 1)
```

Arguments

vcf	A Granges object
ref_genome	Reference genome
extension	The number of bases, that's extracted upstream and downstream of the base substitutions. (Default: 1).

Value

Character vector with the context of the base substitutions

See Also

read_vcfs_as_granges,

Examples

```
## See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
vcfs <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

## Load the corresponding reference genome.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library(ref_genome, character.only = TRUE)

## Get the standard context
mut_context <- mut_context(vcfs[[1]], ref_genome)

## Get larger context
mut_context_larger <- mut_context(vcfs[[1]], ref_genome, extension = 2)
```

---

mut_matrix *Make mutation count matrix of 96 trinucleotides*

Description

Make 96 trinucleotide mutation count matrix

Usage

```
mut_matrix(vcf_list, ref_genome, extension = 1)
```

Arguments

vcf_list	GRangesList or GRanges object.
ref_genome	BSgenome reference genome object
extension	The number of bases, that's extracted upstream and downstream of the base substitutions. (Default: 1).

Value

96 mutation count matrix

See Also

read_vcfs_as_granges

Examples

```
## See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
grl <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

## Load the corresponding reference genome.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library(ref_genome, character.only = TRUE)

## Construct a mutation matrix from the loaded VCFs in comparison to the
## ref_genome.
mut_mat <- mut_matrix(vcf_list = grl, ref_genome = ref_genome)

## Construct a mutation matrix with a larger context.
## This is most usefull when you have many mutations per sample.
mut_mat_extended <- mut_matrix(vcf_list = grl, ref_genome = ref_genome, extension = 2)
```

---

mut_matrix_stranded *Make mutation count matrix of 96 trinucleotides with strand information*

Description

Make a mutation count matrix with 192 features: 96 trinucleotides and 2 strands, these can be transcription or replication strand

Usage

```
mut_matrix_stranded(
  vcf_list,
  ref_genome,
  ranges,
  mode = "transcription",
  extension = 1
)
```

Arguments

vcf_list	GRangesList or GRanges object.
ref_genome	BSgenome reference genome object
ranges	GRanges object with the genomic ranges of: 1. (transcription mode) the gene bodies with strand (+/-) information, or 2. (replication mode) the replication strand with 'strand_info' metadata
mode	"transcription" or "replication", default = "transcription"
extension	The number of bases, that's extracted upstream and downstream of the base substitutions. (Default: 1).

Value

192 mutation count matrix (96 X 2 strands)

See Also

read_vcfs_as_granges, mut_matrix, mut_strand

Examples

```
## See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
grl <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

## Load the corresponding reference genome.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library(ref_genome, character.only = TRUE)

## Transcription strand analysis:
## You can obtain the known genes from the UCSC hg19 dataset using
## Bioconductor:
# BiocManager::install("TxDb.Hsapiens.UCSC.hg19.knownGene")
library("TxDb.Hsapiens.UCSC.hg19.knownGene")
genes_hg19 <- genes(TxDb.Hsapiens.UCSC.hg19.knownGene)

mut_mat_s <- mut_matrix_stranded(grl, ref_genome, genes_hg19,
  mode = "transcription"
)

## You can also use a longer context
mut_mat_s <- mut_matrix_stranded(grl, ref_genome, genes_hg19,
  mode = "transcription", extension = 2
)

## Replication strand analysis:
## Read example bed file with replication direction annotation
repli_file <- system.file("extdata/ReplicationDirectionRegions.bed",
  package = "MutationalPatterns"
```

```
)
repli_strand <- read.table(repli_file, header = TRUE)
repli_strand_granges <- GRanges(
  seqnames = repli_strand$Chr,
  ranges = IRanges(
    start = repli_strand$Start + 1,
    end = repli_strand$Stop
  ),
  strand_info = repli_strand$Class
)
## UCSC seqlevelsstyle
seqlevelsStyle(repli_strand_granges) <- "UCSC"
# The levels determine the order in which the features
# will be countend and plotted in the downstream analyses
# You can specify your preferred order of the levels:
repli_strand_granges$strand_info <- factor(
  repli_strand_granges$strand_info,
  levels = c("left", "right")
)

mut_mat_s_rep <- mut_matrix_stranded(grl, ref_genome, repli_strand_granges,
  mode = "replication"
)
```

`mut_strand` *Find strand of mutations*

Description

Find strand of mutations

Usage

```
mut_strand(vcf, ranges, mode = "transcription")
```

Arguments

`vcf`	GRanges containing the VCF object
`ranges`	GRanges object with the genomic ranges of: 1. (transcription mode) the gene bodies with strand (+/-) information, or 2. (replication mode) the replication strand with 'strand_info' metadata
`mode`	"transcription" or "replication", default = "transcription"

Details

For transcription mode: Definitions of gene bodies with strand (+/-) information should be defined in a GRanges object.

For the base substitutions that are within gene bodies, it is determined whether the "C" or "T" base is on the same strand as the gene definition. (Since by convention we regard base substitutions as C>X or T>X.)

Base substitutions on the same strand as the gene definitions are considered "untranscribed", and on the opposite strand of gene bodies as "transcribed", since the gene definitions report the coding or sense strand, which is untranscribed.

No strand information "-" is returned for base substitutions outside gene bodies, or base substitutions that overlap with more than one gene body on the same strand.

For replication mode: Replication directions of genomic ranges should be defined in GRanges object. The GRanges object should have a "strand_info" metadata column, which contains only two different annotations, e.g. "left" and "right", or "leading" and "lagging". The genomic ranges cannot overlap, to allow only one annotation per location.

For each base substitution it is determined on which strand it is located. No strand information "-" is returned for base substitutions in unannotated genomic regions.

With the package we provide an example dataset, see example code.

Value

Character vector with transcriptional strand information with length of vcf: "-" for positions outside gene bodies, "U" for untranscribed/sense/coding strand, "T" for transcribed/anti-sense/non-coding strand.

See Also

read_vcfs_as_granges,

Examples

```
## For this example we need our variants from the VCF samples, and
## a known genes dataset.  See the 'read_vcfs_as_granges()' example
## for how to load the VCF samples.
vcfs <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

## For transcription strand:
## You can obtain the known genes from the UCSC hg19 dataset using
## Bioconductor:
# source("https://bioconductor.org/biocLite.R")
# biocLite("TxDb.Hsapiens.UCSC.hg19.knownGene")
library("TxDb.Hsapiens.UCSC.hg19.knownGene")
genes_hg19 <- genes(TxDb.Hsapiens.UCSC.hg19.knownGene)

mut_strand(vcfs[[1]], genes_hg19, mode = "transcription")

## For replication strand:
## Read example bed file with replication direction annotation
## Read replistrand data
repli_file <- system.file("extdata/ReplicationDirectionRegions.bed",
```

```
  package = "MutationalPatterns"
)
repli_strand <- read.table(repli_file, header = TRUE)
repli_strand_granges <- GRanges(
  seqnames = repli_strand$Chr,
  ranges = IRanges(
    start = repli_strand$Start + 1,
    end = repli_strand$Stop
  ),
  strand_info = repli_strand$Class
)
## UCSC seqlevelsstyle
seqlevelsStyle(repli_strand_granges) <- "UCSC"

mut_strand(vcfs[[1]], repli_strand_granges, mode = "transcription")
```

`mut_type` *Retrieve base substitution types from a VCF object*

Description

A function to extract the base substitutions from a vcf and translate to the 6 common base substitution types.

Usage

```
mut_type(vcf)
```

Arguments

`vcf`	A CollapsedVCF object

Value

Character vector with base substitution types

See Also

`read_vcfs_as_granges`

Examples

```
## See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
vcfs <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

mut_type(vcfs[[1]])
```

mut_type_occurrences *Count the occurrences of each base substitution type*

Description

Count the occurrences of each base substitution type

Usage

```
mut_type_occurrences(vcf_list, ref_genome)
```

Arguments

vcf_list	GRangesList or GRanges object.
ref_genome	BSgenome reference genome object

Value

data.frame with counts of each base substitution type for each sample in vcf_list

See Also

read_vcfs_as_granges,

Examples

```
## See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
vcfs <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

## Load a reference genome.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library(ref_genome, character.only = TRUE)

## Get the type occurrences for all VCF objects.
type_occurrences <- mut_type_occurrences(vcfs, ref_genome)
```

plot_192_profile *Plot 192 trinucleotide profile*

Description

Plot relative contribution of 192 trinucleotides

Usage

```
plot_192_profile(mut_matrix, colors = NA, ymax = 0.2, condensed = FALSE)
```

Arguments

mut_matrix	192 trinucleotide profile matrix
colors	6 value color vector
ymax	Y axis maximum value, default = 0.2
condensed	More condensed plotting format. Default = F.

Value

192 trinucleotide profile plot

See Also

mut_matrix_stranded, extract_signatures, plot_96_profile

Examples

```
## See the 'mut_matrix_stranded()' example for how we obtained the
## mutation matrix with transcriptional strand information:
mut_mat_s <- readRDS(system.file("states/mut_mat_s_data.rds",
  package = "MutationalPatterns"
))

## Plot profile for some of the samples
plot_192_profile(mut_mat_s[, c(1, 4, 7)])

## You can create a more condensed version of the plot
plot_192_profile(mut_mat_s[, c(1, 4, 7)], condensed = TRUE)

## It's also possible to plot signatures, for example signatures
## generated with NMF
## See 'extract_signatures()' on how we obtained these signatures.
nmf_res_strand <- readRDS(system.file("states/nmf_res_strand_data.rds",
  package = "MutationalPatterns"
))

## Optionally, provide signature names
```

```
colnames(nmf_res_strand$signatures) <- c("Signature A", "Signature B")

## Generate the plot
plot_192_profile(nmf_res_strand$signatures)
```

`plot_96_profile` *Plot 96 trinucleotide profile*

Description

Plot relative contribution of 96 trinucleotides

Usage

```
plot_96_profile(mut_matrix, colors = NA, ymax = 0.2, condensed = FALSE)
```

Arguments

`mut_matrix`	96 trinucleotide profile matrix
`colors`	Optional 6 value color vector.
`ymax`	Y axis maximum value, default = 0.2
`condensed`	More condensed plotting format. Default = F.

Value

96 trinucleotide profile plot

See Also

`mut_matrix`, `plot_profile_heatmap`, `plot_river`

Examples

```
## See the 'mut_matrix()' example for how we obtained the
## mutation matrix information:
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))

## Plot the 96-profile of three samples
plot_96_profile(mut_mat[, c(1, 4, 7)])

## Plot a condensed profile
plot_96_profile(mut_mat[, c(1, 4, 7)], condensed = TRUE)

## It's also possible to plot signatures, for example signatures
## generated with NMF
## See 'extract_signatures()' on how we obtained these signatures.
```

```
nmf_res <- readRDS(system.file("states/nmf_res_data.rds",
  package = "MutationalPatterns"
))

## Optionally, provide signature names
colnames(nmf_res$signatures) <- c("Signature A", "Signature B")

## Generate the plot
plot_96_profile(nmf_res$signatures)
```

`plot_bootstrapped_contribution` *Plot the bootstrapped signature contributions*

Description

Plot the signature contributions retrieved with 'fit_to_signatures_bootstrapped'. The function can plot both the absolute or the relative signature contribution. The graph can be plotted as either a jitter plot or as a barplot.

Usage

```
plot_bootstrapped_contribution(
  contri_boots,
  mode = c("absolute", "relative"),
  plot_type = c("jitter", "barplot", "dotplot")
)
```

Arguments

`contri_boots`	matrix showing signature contributions across bootstrap iterations.
`mode`	Either "absolute" for absolute number of mutations, or "relative" for relative contribution, default = "absolute"
`plot_type`	Either "jitter" for a jitter plot, "barplot" for a barplot, or "dotplot" for a dotplot

Value

A ggplot2 graph

Examples

```
## Get the bootstrapped signature contributions
## See 'count_indel_contexts()' for more info on how to do this.
contri_boots <- readRDS(system.file("states/bootstrapped_snv_refit.rds",
  package = "MutationalPatterns"
))

## Plot bootstrapped contribution
```

```
plot_bootstrapped_contribution(contri_boots)

## Plot bootstrapped contribution with relative contributions
plot_bootstrapped_contribution(contri_boots, mode = "relative")

## Plot bootstrapped contribution with a barplot
plot_bootstrapped_contribution(contri_boots, plot_type = "barplot")

## Plot bootstrapped contribution with a dotplot
plot_bootstrapped_contribution(contri_boots, plot_type = "dotplot", mode = "absolute")
```

`plot_compare_dbs` *Compare two DBS mutation profiles*

Description

Plots two DBS mutation profiles and their difference, reports the residual sum of squares (RSS).

Usage

```
plot_compare_dbs(
  profile1,
  profile2,
  profile_names = c("profile 1", "profile 2"),
  profile_ymax = 0.2,
  diff_ylim = c(-0.1, 0.1)
)
```

Arguments

`profile1`	First mutation profile
`profile2`	Second mutation profile
`profile_names`	Character vector with names of the mutations profiles used for plotting, default = c("profile 1", "profile 2")
`profile_ymax`	Maximum value of y-axis (relative contribution) for profile plotting. This can only be used to increase the y axis. If bars fall outside this limit, the maximum value is automatically increased. default = 0.2.
`diff_ylim`	Y-axis limits for profile difference plot, default = c(-0.1, 0.1)

Value

A ggplot2 object

See Also

`plot_compare_profiles`, `plot_compare_indels`, `plot_compare_mbs`

Other DBS: `count_dbs_contexts()`, `get_dbs_context()`, `plot_dbs_contexts()`, `plot_main_dbs_contexts()`

Examples

```
## Get the DBS counts
## See 'count_dbs_contexts()' for more info on how to do this.
dbs_counts <- readRDS(system.file("states/blood_dbs_counts.rds",
  package = "MutationalPatterns"
))

## Get DBS refit info.
## See 'fit_to_signatures()' for more info on how to do this.
fit_res <- readRDS(system.file("states/dbs_refit.rds",
  package = "MutationalPatterns"
))

## Compare the reconstructed profile of sample 1 with the original profile
## The same thing could be done with a reconstructed profile from NMF.
plot_compare_dbs(dbs_counts[, 1], fit_res$reconstructed[, 1])

## You could also compare regular mutation profiles with eachother.
plot_compare_dbs(
  dbs_counts[, 1],
  dbs_counts[, 2]
)

## Or change the names of the profiles
plot_compare_dbs(dbs_counts[, 1],
  dbs_counts[, 2],
  profile_names = c("Original", "Reconstructed")
)

## You can also change the y limits.
## This can be done separately for the profiles and the different facets.
plot_compare_dbs(dbs_counts[, 1],
  dbs_counts[, 2],
  profile_ymax = 0.3,
  diff_ylim = c(-0.03, 0.03)
)
```

---

plot_compare_indels *Compare two indel mutation profiles*

Description

Plots two indel mutation profiles and their difference, reports the residual sum of squares (RSS).

Usage

```
plot_compare_indels(
  profile1,
  profile2,
```

```
  profile_names = c("profile 1", "profile 2"),
  profile_ymax = 0.2,
  diff_ylim = c(-0.1, 0.1)
)
```

Arguments

profile1	First mutation profile
profile2	Second mutation profile
profile_names	Character vector with names of the mutations profiles used for plotting, default = c("profile 1", "profile 2")
profile_ymax	Maximum value of y-axis (relative contribution) for profile plotting. This can only be used to increase the y axis. If bars fall outside this limit, the maximum value is automatically increased. default = 0.2.
diff_ylim	Y-axis limits for profile difference plot, default = c(-0.1, 0.1)

Value

A ggplot2 object

See Also

plot_compare_profiles, plot_compare_dbs, plot_compare_mbs

Other Indels: count_indel_contexts(), get_indel_context(), plot_indel_contexts(), plot_main_indel_contexts

Examples

```
## Get the indel counts
## See 'count_indel_contexts()' for more info on how to do this.
indel_counts <- readRDS(system.file("states/blood_indel_counts.rds",
  package = "MutationalPatterns"
))

## Get indel refit info.
## See 'fit_to_signatures()' for more info on how to do this.
fit_res <- readRDS(system.file("states/indel_refit.rds",
  package = "MutationalPatterns"
))

## Compare the reconstructed profile of sample 1 with the original profile
## The same thing could be done with a reconstructed profile from NMF.
plot_compare_indels(indel_counts[, 1], fit_res$reconstructed[, 1])

## You could also compare regular mutation profiles with eachother.
plot_compare_indels(
  indel_counts[, 1],
  indel_counts[, 2]
)

## Or change the names of the profiles
```

```
plot_compare_indels(indel_counts[, 1],
  indel_counts[, 2],
  profile_names = c("Original", "Reconstructed")
)

## You can also change the y limits.
## This can be done separately for the profiles and the different facets.
plot_compare_indels(indel_counts[, 1],
  indel_counts[, 2],
  profile_ymax = 0.3,
  diff_ylim = c(-0.03, 0.03)
)
```

`plot_compare_mbs` *Compare two mbs mutation profiles*

Description

Plots two mbs mutation profiles and their difference, reports the residual sum of squares (RSS).

Usage

```
plot_compare_mbs(
  profile1,
  profile2,
  profile_names = c("profile 1", "profile 2"),
  profile_ymax = 1,
  diff_ylim = c(-0.5, 0.5)
)
```

Arguments

`profile1`	First mutation profile
`profile2`	Second mutation profile
`profile_names`	Character vector with names of the mutations profiles used for plotting, default = c("profile 1", "profile 2")
`profile_ymax`	Maximum value of y-axis (relative contribution) for profile plotting. This can only be used to increase the y axis. If bars fall outside this limit, the maximum value is automatically increased. default = 1.
`diff_ylim`	Y-axis limits for profile difference plot, default = c(-0.5, 0.5)

Value

A ggplot2 object

See Also

plot_compare_profiles, plot_compare_dbs, plot_compare_indels

Other MBS: count_mbs_contexts(), plot_mbs_contexts()

Examples

```
## Get the mbs counts
## See 'count_mbs_contexts()' for more info on how to do this.
mbs_counts <- readRDS(system.file("states/blood_mbs_counts.rds",
  package = "MutationalPatterns"
))


## You could compare regular mutation profiles with eachother.
plot_compare_mbs(
  mbs_counts[, 1],
  mbs_counts[, 2]
)

## Or change the names of the profiles
plot_compare_mbs(mbs_counts[, 1],
  mbs_counts[, 2],
  profile_names = c("Original", "Reconstructed")
)

## You can also change the y limits.
## This can be done separately for the profiles and the different facets.
plot_compare_mbs(mbs_counts[, 1],
  mbs_counts[, 2],
  profile_ymax = 0.9,
  diff_ylim = c(-0.8, 0.8)
)

## You could also compare a reconstructed profile.
## However, the example data does not contain enough MBS variants to use NMF.
## Existing signatures have also not yet been defined.
```

---

plot_compare_profiles *Compare two 96 mutation profiles*

Description

Plots two 96 mutation profiles and their difference, reports the residual sum of squares (RSS).

Usage

```
plot_compare_profiles(
  profile1,
  profile2,
```

```
  profile_names = c("profile 1", "profile 2"),
  profile_ymax = 0.2,
  diff_ylim = c(-0.02, 0.02),
  colors = NA,
  condensed = FALSE
)
```

Arguments

profile1	First 96 mutation profile
profile2	Second 96 mutation profile
profile_names	Character vector with names of the mutations profiles used for plotting, default = c("profile 1", "profile 2")
profile_ymax	Maximum value of y-axis (relative contribution) for profile plotting. This can only be used to increase the y axis. If bars fall outside this limit, the maximum value is automatically increased. default = 0.2.
diff_ylim	Y-axis limits for profile difference plot, default = c(-0.02, 0.02)
colors	6 value color vector
condensed	More condensed plotting format. Default = F.

Value

96 spectrum plot of profile 1, profile 2 and their difference

See Also

mut_matrix, extract_signatures, plot_compare_indels, plot_compare_dbs, plot_compare_mbs

Examples

```
## See the 'mut_matrix()' example for how we obtained the following
## mutation matrix.
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))

## Extracting signatures can be computationally intensive, so
## we use pre-computed data generated with the following command:
# nmf_res <- extract_signatures(mut_mat, rank = 2)

nmf_res <- readRDS(system.file("states/nmf_res_data.rds",
  package = "MutationalPatterns"
))

## Compare the reconstructed 96-profile of sample 1 with the original profile
## The same thing could be done with a reconstructed profile from signature refitting.
plot_compare_profiles(mut_mat[, 1],
  nmf_res$reconstructed[, 1],
  profile_names = c("Original", "Reconstructed")
```

```
)

## You could also compare regular mutation profiles with eachother.
plot_compare_profiles(
  mut_mat[, 1],
  mut_mat[, 2]
)


## You can also change the y limits.
## This can be done separately for the profiles and the different facets.
plot_compare_profiles(mut_mat[, 1],
  mut_mat[, 2],
  profile_ymax = 0.3,
  diff_ylim = c(-0.03, 0.03)
)
```

`plot_contribution` *Plot signature contribution barplot*

Description

Plot contribution of signatures. Can be used on both the results of a NMF and on the results of signature refitting.

Usage

```
plot_contribution(
  contribution,
  signatures = NA,
  index = NA,
  coord_flip = FALSE,
  mode = c("relative", "absolute"),
  palette = NA
)
```

Arguments

`contribution`	Signature contribution matrix
`signatures`	Signature matrix. Necessary when plotting NMF results in "absolute" mode. It's not necessary in relative mode or when visualizing signature refitting results
`index`	optional sample subset parameter
`coord_flip`	Flip X and Y coordinates, default = FALSE
`mode`	"relative" or "absolute"; to plot the relative contribution or absolute number of mutations, default = "relative"
`palette`	A color palette like c("#FF0000", "#00FF00", "9999CC") that will be used as colors in the plot. By default, ggplot2's colors are used to generate a palette.

Value

Stacked barplot with contribution of each signature for each sample

See Also

extract_signatures, mut_matrix

Examples

```
## Extracting signatures can be computationally intensive, so
## we use pre-computed data generated with the following command:
# nmf_res <- extract_signatures(mut_mat, rank = 2)

nmf_res <- readRDS(system.file("states/nmf_res_data.rds",
  package = "MutationalPatterns"
))

## Optionally set column and row names.
colnames(nmf_res$signatures) <- c("Signature A", "Signature B")
rownames(nmf_res$contribution) <- c("Signature A", "Signature B")

## Plot the relative contribution
plot_contribution(nmf_res$contribution)

## Plot the absolute contribution.
## When plotting absolute NMF results, the signatures need to be included.
plot_contribution(nmf_res$contribution,
  nmf_res$signature,
  mode = "absolute"
)


## Only plot a subset of samples
plot_contribution(nmf_res$contribution,
  nmf_res$signature,
  mode = "absolute",
  index = c(1, 2)
)
## Flip the coordinates
plot_contribution(nmf_res$contribution,
  nmf_res$signature,
  mode = "absolute",
  coord_flip = TRUE
)

## You can also use the results of signature refitting.
## Here we load some data as an example
fit_res <- readRDS(system.file("states/snv_refit.rds",
  package = "MutationalPatterns"
))
plot_contribution(fit_res$contribution)
```

```
## Or again in absolute mode
plot_contribution(fit_res$contribution,
  mode = "absolute"
)
```

`plot_contribution_heatmap` *Plot signature contribution heatmap*

Description

Plot relative contribution of signatures in a heatmap

Usage

```
plot_contribution_heatmap(
  contribution,
  sig_order = NA,
  sample_order = NA,
  cluster_samples = TRUE,
  cluster_sigs = FALSE,
  method = "complete",
  plot_values = FALSE
)
```

Arguments

`contribution`	Signature contribution matrix
`sig_order`	Character vector with the desired order of the signature names for plotting. Optional.
`sample_order`	Character vector with the desired order of the sample names for plotting. Optional.
`cluster_samples`	Hierarchically cluster samples based on euclidean distance. Default = T.
`cluster_sigs`	Hierarchically cluster sigs based on euclidean distance. Default = T.
`method`	The agglomeration method to be used for hierarchical clustering. This should be one of "ward.D", "ward.D2", "single", "complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC) or "centroid" (= UPGMC). Default = "complete".
`plot_values`	Plot relative contribution values in heatmap. Default = F.

Value

Heatmap with relative contribution of each signature for each sample

See Also

extract_signatures, mut_matrix, plot_contribution, plot_cosine_heatmap

Examples

```
## Extracting signatures can be computationally intensive, so
## we use pre-computed data generated with the following command:
# nmf_res <- extract_signatures(mut_mat, rank = 2)

nmf_res <- readRDS(system.file("states/nmf_res_data.rds",
  package = "MutationalPatterns"
))

## Set signature names as row names in the contribution matrix
rownames(nmf_res$contribution) <- c("Signature A", "Signature B")

## Plot with clustering.
plot_contribution_heatmap(nmf_res$contribution, cluster_samples = TRUE, cluster_sigs = TRUE)

## Define signature and sample order for plotting. If you have a mutation or signature
## matrix, then this can be done like in the example of 'plot_cosine_heatmap()'
sig_order <- c("Signature B", "Signature A")
sample_order <- c(
  "colon1", "colon2", "colon3", "intestine1", "intestine2",
  "intestine3", "liver3", "liver2", "liver1"
)
plot_contribution_heatmap(nmf_res$contribution,
  cluster_samples = FALSE,
  sig_order = sig_order, sample_order = sample_order
)

## It's also possible to create a contribution heatmap with text values
output_text <- plot_contribution_heatmap(nmf_res$contribution, plot_values = TRUE)

## This function can also be used on the result of a signature refitting analysis.
## Here we load a existing result as an example.
snv_refit <- readRDS(system.file("states/strict_snv_refit.rds",
  package = "MutationalPatterns"
))
plot_contribution_heatmap(snv_refit$contribution, cluster_samples = TRUE, cluster_sigs = TRUE)
```

---

plot_correlation_bootstrap

Plots the correlation between bootstrapped signature contributions

---

Description

This function plots the pearson correlation between signatures. This can be done per sample or for all samples together. It returns a list of the created figures.

Usage

```
plot_correlation_bootstrap(contri_boots, per_sample = TRUE)
```

Arguments

contri_boots	A dataframe with bootstrapped signature contributions.
per_sample	Whether or not a plot should be made per sample. Default: TRUE.

Value

A list of ggplot2 objects if run per sample. Else it returns a single ggplot2 object.

Examples

```
## Get a dataframe with bootstrapped signature contributions.
## See 'fit_to_signatures_bootstrapped()' for how to do this.
contri_boots <- readRDS(system.file("states/bootstrapped_snv_refit.rds",
  package = "MutationalPatterns"
))

## Plot the correlations between signatures per sample
fig_l <- plot_correlation_bootstrap(contri_boots)

## Look at the figure of the first sample.
fig_l[[1]]

## You can also look at the correlation for all samples combined
plot_correlation_bootstrap(contri_boots, per_sample = FALSE)
```

---

plot_cosine_heatmap *Plot cosine similarity heatmap*

Description

Plot pairwise cosine similarities in a heatmap.

Usage

```
plot_cosine_heatmap(
  cos_sim_matrix,
  col_order = NA,
  row_order = NA,
  cluster_rows = TRUE,
  cluster_cols = FALSE,
  method = "complete",
  plot_values = FALSE
)
```

Arguments

cos_sim_matrix	Matrix with pairwise cosine similarities. Result from cos_sim_matrix
col_order	Character vector with the desired order of the columns names for plotting. Optional.
row_order	Character vector with the desired order of the row names for plotting. Optional.
cluster_rows	Hierarchically cluster rows based on euclidean distance. Default = TRUE.
cluster_cols	Hierarchically cluster cols based on euclidean distance. Default = FALSE.
method	The agglomeration method to be used for hierarchical clustering. This should be one of "ward.D", "ward.D2", "single", "complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC) or "centroid" (= UPGMC). Default = "complete".
plot_values	Plot cosine similarity values in heatmap. Default = FALSE.

Value

Heatmap with cosine similarities

See Also

mut_matrix, cos_sim_matrix

Examples

```
## See the 'mut_matrix()' example for how we obtained the mutation matrix:
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))

## Get signatures
signatures <- get_known_signatures()

## Calculate the cosine similarity between each signature and each 96 mutational profile
cos_matrix <- cos_sim_matrix(mut_mat, signatures)


## Plot the cosine similarity between each signature and each sample with hierarchical
## clustering of samples and signatures.
plot_cosine_heatmap(cos_matrix, cluster_rows = TRUE, cluster_cols = TRUE)

## In the above example, clustering is performed on the similarities of the samples with
## the signatures. It's also possible to cluster the signatures and samples on their (96) profile.
## This will generally give better results
## If you use the same signatures for different analyses,
## then their order will also be consistent.
hclust_cosmic <- cluster_signatures(signatures, method = "average")
cosmic_order <- colnames(signatures)[hclust_cosmic$order]
hclust_samples <- cluster_signatures(mut_mat, method = "average")
sample_order <- colnames(mut_mat)[hclust_samples$order]
## Plot the cosine heatmap using this given signature order.
```

```
plot_cosine_heatmap(cos_matrix,
  cluster_rows = FALSE, cluster_cols = FALSE,
  row_order = sample_order, col_order = cosmic_order
)

## You can also plot the similarity of samples with eachother
cos_matrix <- cos_sim_matrix(mut_mat, mut_mat)
plot_cosine_heatmap(cos_matrix, cluster_rows = TRUE, cluster_cols = TRUE)


## It's also possible to add the actual values in the heatmap.
plot_cosine_heatmap(cos_matrix, cluster_rows = TRUE, cluster_cols = TRUE, plot_values = TRUE)
```

`plot_dbs_contexts` *Plot the DBS contexts*

Description

Plot the DBS contexts

Usage

```
plot_dbs_contexts(counts, same_y = FALSE, condensed = FALSE)
```

Arguments

`counts`	A tibble containing the number of DBS per COSMIC context.
`same_y`	A boolean describing whether the same y axis should be used for all samples.
`condensed`	More condensed plotting format. Default = F.

Details

Plots the number of DBS COSMIC context per sample. It takes a tibble with counts as its input. This tibble can be generated by count_dbs_contexts Each sample is plotted in a separate facet. The same y axis can be used for all samples or a separate y axis can be used.

Value

A ggplot figure.

See Also

`count_dbs_contexts`, `plot_main_dbs_contexts`

Other DBS: `count_dbs_contexts()`, `get_dbs_context()`, `plot_compare_dbs()`, `plot_main_dbs_contexts()`

Examples

```
## Get The DBS counts
## See 'count_dbs_contexts()' for more info on how to do this.
dbs_counts <- readRDS(system.file("states/blood_dbs_counts.rds",
  package = "MutationalPatterns"
))

## Plot contexts
plot_dbs_contexts(dbs_counts)

## Use the same y axis for all samples.
plot_dbs_contexts(dbs_counts, same_y = TRUE)

## Create a more condensed plot
plot_dbs_contexts(dbs_counts, condensed = TRUE)
```

---

plot_enrichment_depletion

Plot enrichment/depletion of mutations in genomic regions

Description

Plot enrichment/depletion of mutations in genomic regions

Usage

```
plot_enrichment_depletion(df, sig_type = c("fdr", "p"))
```

Arguments

df	Dataframe result from enrichment_depletion_test()
sig_type	The type of significance to be used. Possible values: * 'fdr' False discovery rate. A type of multiple testing correction.; * 'p' for regular p values.

Value

Plot with two parts. 1: Barplot with no. mutations expected and observed per region. 2: Effect size of enrichment/depletion (log2ratio) with results significance test.

See Also

enrichment_depletion_test, genomic_distribution

Examples

```
## See the 'genomic_distribution()' example for how we obtained the
## following data:
distr <- readRDS(system.file("states/distr_data.rds",
  package = "MutationalPatterns"
))

tissue <- c(
  "colon", "colon", "colon",
  "intestine", "intestine", "intestine",
  "liver", "liver", "liver"
)

## Perform the enrichment/depletion test.
distr_test <- enrichment_depletion_test(distr, by = tissue)

## Plot the enrichment/depletion
plot_enrichment_depletion(distr_test)

#Perform and plot the enrichmet depletion test for all samples pooled
distr_test2 <- enrichment_depletion_test(distr)
plot_enrichment_depletion(distr_test2)

## Plot with p values instead of fdr
plot_enrichment_depletion(distr_test, sig_type = "p")

## Use multiple (max 3) significance cutoffs.
## This will vary the number of significance stars.
distr_multistars <- enrichment_depletion_test(distr,
  by = tissue,
  p_cutoffs = c(0.05, 0.01, 0.005),
  fdr_cutoffs = c(0.1, 0.05, 0.01)
)
plot_enrichment_depletion(distr_multistars)
```

---

plot_indel_contexts *Plot the indel contexts*

Description

Plot the indel contexts

Usage

```
plot_indel_contexts(
  counts,
  same_y = FALSE,
  extra_labels = FALSE,
  condensed = FALSE
)
```

Arguments

counts	A tibble containing the number of indels per COSMIC context.
same_y	A boolean describing whether the same y axis should be used for all samples.
extra_labels	A boolean describing whether extra labels should be added. These can clarify the plot, but will shift when different plot widths are used. We recommend saving a plot with a width of 12, when using this argument.
condensed	More condensed plotting format. Default = F.

Details

Plots the number of indels COSMIC context per sample. It takes a tibble with counts as its input. This tibble can be generated by 'count_indel_contexts()'. Each sample is plotted in a separate facet. The same y axis can be used for all samples or a separate y axis can be used. The facets at the top show the indel types. First the C and T deletions Then the C and T insertions. Next are the multi base deletions and insertions. Finally the deletions with microhomology (mh) are shown. The x-axis at the bottom shows the number of repeat units. For mh deletions the microhomology length is shown.

Value

A ggplot figure.

See Also

count_indel_contexts, plot_main_indel_contexts

Other Indels: count_indel_contexts(), get_indel_context(), plot_compare_indels(), plot_main_indel_contexts

Examples

```
## Get The indel counts
## See 'count_indel_contexts()' for more info on how to do this.
indel_counts <- readRDS(system.file("states/blood_indel_counts.rds",
  package = "MutationalPatterns"
))

## Plot contexts
plot_indel_contexts(indel_counts)

## Use the same y axis for all samples.
plot_indel_contexts(indel_counts, same_y = TRUE)

## Add extra labels to make plot clearer
plot_indel_contexts(indel_counts, extra_labels = TRUE)

## Create a more condensed plot
plot_indel_contexts(indel_counts, condensed = TRUE)
```

`plot_lesion_segregation` *Plot the strands of variants to show lesion segregation*

Description

The strands of variants in a GRanges object is plotted. This way the presence of any lesion segregation is visualized. The function can plot either a single or multiple samples. Per chromosome, the ratio of the mutations on the chromosomal strands is visualised by a line. The position of this line is calculated as the mean of the "+" and "-" strand, where "+" equals 1 and "-" equals 0. In other words: this line lies between the two strands if the mutations are equally distributed between them, and approaches a strand if the majority of mutations on a chromosome lie on that strand.

Usage

```
plot_lesion_segregation(
  vcf,
  per_chrom = FALSE,
  sample_name = NA,
  min_muts_mean = 10,
  chromosomes = NA,
  subsample = NA
)
```

Arguments

`vcf`	GRanges or RGrangesList object.
`per_chrom`	Boolean. Determines whether to create a separate plot per chromosome.
`sample_name`	Name of the sample. Is used as the title of the plot. Not very useful if you have more than one sample.
`min_muts_mean`	Integer. The minimum of mutations, required for the mean strand of a chromosome to be calculated.
`chromosomes`	Character vector. Determines chromosomes to be used and their order.
`subsample`	Double between 0 and 1. Subsamples the amount of mutations to create a plot with less dots. Such a plot is easier to modify in a vector program like illustrator. (default: NA)

Value

ggplot2 object

See Also

`calculate_lesion_segregation`

Other Lesion_segregation: `calculate_lesion_segregation()`

Examples

```
## See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
grl <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

## Plot lesion segregation
plot_lesion_segregation(grl[1:3])

## Select a single GRanges object to plot.
gr <- grl[[1]]

## Plot lesion segregation for a single sample.
## Also add a title to the plot.
plot_lesion_segregation(gr, sample_name = "Colon1")

## Plot lesion segregation per chromosome.
## We here store the results in a list.
figure_l = plot_lesion_segregation(gr, per_chrom = TRUE, sample_name = "Colon1")

## Plot specific chromosomes in a user specified order
plot_lesion_segregation(grl[1:3], chromosomes = c(2,3))

## Subsample the mutations, so less points are plotted.
plot_lesion_segregation(grl[1:3], subsample = 0.2)
```

---

plot_main_dbs_contexts

Plot the main DBS contexts

Description

Plot the main DBS contexts

Usage

```
plot_main_dbs_contexts(counts, same_y = FALSE)
```

Arguments

counts	A tibble containing the number of DBS per COSMIC context.
same_y	A boolean describing whether the same y axis should be used for all samples.

Details

Plots the number of DBS per main COSMIC context per sample. The contexts are only divided by REF and not by ALT. It takes a tibble with counts as its input. This tibble can be generated by count_dbs_contexts Each sample is plotted in a separate facet. The same y axis can be used for all samples or a separate y axis can be used.

Value

A ggplot figure.

See Also

count_dbs_contexts, plot_dbs_contexts

Other DBS: count_dbs_contexts(), get_dbs_context(), plot_compare_dbs(), plot_dbs_contexts()

Examples

```
## Get The DBS counts
## See 'count_dbs_contexts()' for more info on how to do this.
dbs_counts <- readRDS(system.file("states/blood_dbs_counts.rds",
  package = "MutationalPatterns"
))

## Plot contexts
plot_main_dbs_contexts(dbs_counts)

## Use the same y axis for all samples.
plot_main_dbs_contexts(dbs_counts, same_y = TRUE)
```

---

plot_main_indel_contexts

Plot the main indel contexts

Description

Plot the main indel contexts

Usage

```
plot_main_indel_contexts(counts, same_y = FALSE)
```

Arguments

counts	A tibble containing the number of indels per COSMIC context.
same_y	A boolean describing whether the same y axis should be used for all samples.

Details

Plots the number of indels per main COSMIC context per sample. The contexts are not subdivided into the number of repeats/microhomology length. It takes a tibble with counts as its input. This tibble can be generated by count_indel_contexts Each sample is plotted in a separate facet. The same y axis can be used for all samples or a separate y axis can be used.

Value

A ggplot figure.

See Also

count_indel_contexts, plot_indel_contexts

Other Indels: count_indel_contexts(), get_indel_context(), plot_compare_indels(), plot_indel_contexts()

Examples

```
## Get The indel counts
## See 'count_indel_contexts()' for more info on how to do this.
indel_counts <- readRDS(system.file("states/blood_indel_counts.rds",
  package = "MutationalPatterns"
))

## Plot contexts
plot_main_indel_contexts(indel_counts)

## Use the same y axis for all samples.
plot_main_indel_contexts(indel_counts, same_y = TRUE)
```

---

plot_mbs_contexts *Plot the MBS contexts*

Description

Plot the MBS contexts

Usage

```
plot_mbs_contexts(counts, same_y = TRUE)
```

Arguments

counts	A tibble containing the number of MBS per MBS length.
same_y	A boolean describing whether the same y axis should be used for all samples.

Details

Plots the number of MBS per MBS length per sample. It takes a tibble with counts as its input. This tibble can be generated by count_mbs_contexts Each sample is plotted in a separate facet. The same y axis can be used for all samples or a separate y axis can be used.

Value

A ggplot figure.

See Also

count_mbs_contexts

Other MBS: count_mbs_contexts(), plot_compare_mbs()

Examples

```
## Get The mbs counts
## See 'count_mbs_contexts()' for more info on how to do this.
mbs_counts <- readRDS(system.file("states/blood_mbs_counts.rds",
  package = "MutationalPatterns"
))

## Plot contexts
plot_mbs_contexts(mbs_counts)

## Use a different y axis for all samples.
plot_mbs_contexts(mbs_counts, same_y = FALSE)
```

---

plot_original_vs_reconstructed

Plot the similarity between a mutation matrix and its reconstructed profile

---

Description

When a reconstructed profile has a cosine similarity of more than 0.95 with original, the reconstructed profile is considered very good.

Usage

```
plot_original_vs_reconstructed(
  mut_matrix,
  reconstructed,
  y_intercept = 0.95,
  ylims = c(0.6, 1)
)
```

Arguments

mut_matrix	mutation count matrix (dimensions: x mutation types X n samples)
reconstructed	A reconstructed mutation count matrix
y_intercept	The y intercept of the plotted horizontal line. Default: 0.95.
ylims	The limits of the y axis. Default: c(0.6, 1)

Value

A ggplot figure

Examples

```
## See the 'mut_matrix()' example for how we obtained the mutation matrix:
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))

## Extracting signatures can be computationally intensive, so
## we use pre-computed data generated with the following command:
# nmf_res <- extract_signatures(mut_mat, rank = 2)

nmf_res <- readRDS(system.file("states/nmf_res_data.rds",
  package = "MutationalPatterns"
))

## Create figure
plot_original_vs_reconstructed(mut_mat, nmf_res$reconstructed)

## You can also use the results of signature refitting.
## Here we load some data as an example
fit_res <- readRDS(system.file("states/snv_refit.rds",
  package = "MutationalPatterns"
))
plot_original_vs_reconstructed(mut_mat, fit_res$reconstructed)

## You can also change the height of the horizontal line
plot_original_vs_reconstructed(mut_mat, fit_res$reconstructed, y_intercept = 0.90)

## It's also possible to change the limits of the y axis
plot_original_vs_reconstructed(mut_mat, fit_res$reconstructed, ylims = c(0, 1))
```

---

plot_profile_heatmap *Plot a mutation matrix as a heatmap*

Description

Function to plot a SNV mutation matrix as a heatmap. This is especially useful when looking at a wide mutational context.

Usage

```
plot_profile_heatmap(mut_matrix, by = NA, max = 0.02, condensed = FALSE)
```

Arguments

mut_matrix	Matrix containing mutation counts.
by	Optional grouping variable
max	Maximum value used for plotting the relative contributions. Contributions that are higher will have the maximum colour. (Default: 0.02)
condensed	More condensed plotting format. Default = F.

Value

A ggplot object

See Also

mut_matrix, plot_96_profile, plot_river

Examples

```
## See the 'mut_matrix()' examples for how we obtained the
## mutation matrix information:
## Get regular matrix
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))

## Create heatmap of profile
plot_profile_heatmap(mut_mat, max = 0.1)

## Get extended matrix
mut_mat_extended <- readRDS(system.file("states/mut_mat_data_extended.rds",
  package = "MutationalPatterns"
))

## Create heatmap of extended profile
plot_profile_heatmap(mut_mat_extended)

## Or plot heatmap per tissue
tissue <- c(
  "colon", "colon", "colon",
  "intestine", "intestine", "intestine",
  "liver", "liver", "liver"
)

plot_profile_heatmap(mut_mat_extended, by = tissue)

## Or plot the heatmap per sample.
plot_profile_heatmap(mut_mat_extended,
```

```
  by = colnames(mut_mat_extended),
  max = 0.05
)


## Create a condensed heatmap of extended profile
plot_profile_heatmap(mut_mat_extended, condensed = TRUE)
```

`plot_profile_region` *Plot 96 trinucleotide profile per subgroup*

Description

Plot relative contribution of 96 trinucleotides per subgroup. This can be genomic regions but could also be other subsets. The function uses a matrix generated by 'lengthen_mut_matrix()' as its input.

Usage

```
plot_profile_region(
  mut_matrix,
  mode = c("relative_sample", "relative_sample_feature", "absolute"),
  colors = NULL,
  ymax = 0.2,
  condensed = FALSE
)
```

Arguments

`mut_matrix`	Mutation matrix
`mode`	'relative_sample', 'relative_sample_feature' or 'absolute' When 'relative_sample', the number of variants will be shown divided by the total number of variants in that sample. When 'relative_sample_feature', the number of variants will be shown divided by the total number of variants in that sample. and genomic region.
`colors`	6 value color vector
`ymax`	Y axis maximum value, default = 0.2
`condensed`	More condensed plotting format. Default = FALSE.

Value

96 trinucleotide profile plot per region

See Also

`mut_matrix`

Other genomic_regions: `bin_mutation_density()`, `lengthen_mut_matrix()`, `plot_spectrum_region()`, `split_muts_region()`

Examples

```
## See the 'lengthen_mut_matrix()' example for how we obtained the
## mutation matrix information:
mut_mat_long <- readRDS(system.file("states/mut_mat_longregions.rds",
  package = "MutationalPatterns"
))

## Plot the 96-profile of three samples
plot_profile_region(mut_mat_long[, c(1, 4, 7)])
```

`plot_rainfall` *Plot genomic rainfall*

Description

Rainfall plot visualizes the types of mutations and intermutation distance

Usage

```
plot_rainfall(
  vcf,
  chromosomes,
  title = "",
  colors = NA,
  cex = 2.5,
  cex_text = 3,
  ylim = 1e+08,
  type = c("snv", "indel", "dbs", "mbs")
)
```

Arguments

`vcf`	GRanges object
`chromosomes`	Vector of chromosome/contig names of the reference genome to be plotted
`title`	Optional plot title
`colors`	Vector of 6 colors used for plotting
`cex`	Point size
`cex_text`	Text size
`ylim`	Maximum y value (genomic distance)
`type`	The mutation type of the GRanges object that will be used. Possible values: * 'snv' (default) * 'indel' * 'dbs' * 'mbs'

Details

Rainfall plots can be used to visualize the distribution of mutations along the genome or a subset of chromosomes. The distance of a mutation with the mutation prior to it (the intermutation distance) is plotted on the y-axis on a log scale. The input GRanges are sorted before plotting.

The colour of the points indicates the base substitution type. Clusters of mutations with lower intermutation distance represent mutation hotspots.

Value

Rainfall plot

See Also

read_vcfs_as_granges

Examples

```
## See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
vcfs <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

# Specify chromosomes of interest.
chromosomes <- names(genome(vcfs[[1]])[1:22])

## Do a rainfall plot for all chromosomes:
plot_rainfall(vcfs[[1]],
  title = names(vcfs[1]),
  chromosomes = chromosomes,
  cex = 1
)

## Or for a single chromosome (chromosome 1):
plot_rainfall(vcfs[[1]],
  title = names(vcfs[1]),
  chromosomes = chromosomes[1],
  cex = 2
)

## You can also use other variant types

## Get a GRangesList or GRanges object with indel contexts.
## See 'indel_get_context' for more info on how to do this.
grl_indel_context <- readRDS(system.file("states/blood_grl_indel_context.rds",
  package = "MutationalPatterns"
))

plot_rainfall(grl_indel_context[[1]],
  title = "Indel rainfall",
  chromosomes,
```

```
  type = "indel"
)
```

`plot_regional_similarity` *Plot regional similarity*

Description

Plot the cosine similarity of the mutation profiles of small genomic windows with the rest of the genome.

Usage

```
plot_regional_similarity(
  region_cossim,
  per_chrom = FALSE,
  oligo_correction = TRUE,
  max_cossim = NA,
  title = NA,
  plot_rug = FALSE,
  x_axis_breaks = NA
)
```

Arguments

`region_cossim`	A region_cossim object.
`per_chrom`	Boolean. Determines whether to create a separate plot per chromosome. (Default: FALSE)
`oligo_correction`	Boolean describing whether the oligonucleotide frequency corrected cosine similarities should be plotted. If no correction has been applied then the regular cosine similarities will be plotted. (Default: TRUE)
`max_cossim`	Maximum cosine similarity for a window to be considered an outlier. Any window with a lower cosine similarity is given a different color. (Default: NA)
`title`	Optional plot title. (Default: NA). When the default option is used, the number of mutations per window and the step size are shown.
`plot_rug`	Add a bottom rug to the plot, depicting the location of the mutations. (Default: FALSE)
`x_axis_breaks`	Vector of custom x-axis breaks. (Default: NA)

Details

Each dot shows the cosine similarity between the mutation profiles of a single window and the rest of the genome. A region with a different mutation profile will have a lower cosine similarity. The dots are colored based on the sizes in mega bases of the windows. This size is the distance between the first and last mutations in a window. The locations of the mutations can be plotted on the bottom of the figure. The cosine similarity can be plotted both with and without oligonucleotide frequency correction. This can be done for all chromosomes at once or separate plots can be made per chromosome.

Value

ggplot2 object

See Also

determine_regional_similarity

Other regional_similarity: determine_regional_similarity()

Examples

```
## See the 'determine_regional_similarity()' example for how we obtained the
## following data:
regional_sims <- readRDS(system.file("states/regional_sims.rds",
  package = "MutationalPatterns"
))

## Plot the regional similarity
plot_regional_similarity(regional_sims)

## Plot outlier samples with a different color.
## The value of 0.5 that is used here is arbitrarily chosen
## and should in practice be based on the data.
plot_regional_similarity(regional_sims, max_cossim = 0.5)

## Plot samples per chromosome
fig_l = plot_regional_similarity(regional_sims, per_chrom = TRUE)

## Plot without a title
plot_regional_similarity(regional_sims, title = "")

## Add a rug to the plot, that shows the location of the mutations.
plot_regional_similarity(regional_sims, plot_rug = FALSE)

## Use custom x axis breaks
plot_regional_similarity(regional_sims, x_axis_breaks = c(50, 150))
```

plot_river *Plot a riverplot*

Description

Function to plot a SNV mutation matrix as a riverplot. This is especially useful when looking at a wide mutational context

Usage

```
plot_river(mut_matrix, condensed = FALSE)
```

Arguments

mut_matrix	Matrix containing mutation counts.
condensed	More condensed plotting format. Default = F.

Value

A ggplot object

See Also

mut_matrix, plot_96_profile, plot_profile_heatmap

Examples

```
## See the 'mut_matrix()' examples for how we obtained the
## mutation matrix information:
## Get regular matrix
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))

## Create heatmap of profile
plot_river(mut_mat[,c(1,4)])

## Get extended matrix
mut_mat_extended <- readRDS(system.file("states/mut_mat_data_extended.rds",
  package = "MutationalPatterns"
))

## Create heatmap of extended profile
plot_river(mut_mat_extended[,c(1,4)])

## Create condensed version of riverplot
plot_river(mut_mat_extended[,c(1,4)], condensed = TRUE)
```

plot_signature_strand_bias *Plot signature strand bias*

Description

Plot strand bias per mutation type for each signature.

Usage

```
plot_signature_strand_bias(signatures_strand_bias)
```

Arguments

signatures_strand_bias	Signature matrix with 192 features

Value

Barplot

See Also

link{extract_signatures}, link{mut_matrix}

Examples

```
## See the 'mut_matrix()' example for how we obtained the following
## mutation matrix.
mut_mat_s <- readRDS(system.file("states/mut_mat_s_data.rds",
  package = "MutationalPatterns"
))

## Extracting signatures can be computationally intensive, so
## we use pre-computed data generated with the following command:
# nmf_res_strand <- extract_signatures(mut_mat_s, rank = 2)

nmf_res_strand <- readRDS(system.file("states/nmf_res_strand_data.rds",
  package = "MutationalPatterns"
))

## Provide column names for the plot.
colnames(nmf_res_strand$signatures) <- c("Signature A", "Signature B")

## Creat figure
plot_signature_strand_bias(nmf_res_strand$signatures)

## You can also plot the bias of samples
plot_signature_strand_bias(mut_mat_s[, c(1, 2)])
```

plot_spectrum *Plot point mutation spectrum*

Description

Plot point mutation spectrum

Usage

```
plot_spectrum(
  type_occurrences,
  CT = FALSE,
  by = NA,
  indv_points = FALSE,
  error_bars = c("95%_CI", "stdev", "SEM", "none"),
  colors = NA,
  legend = TRUE,
  condensed = FALSE
)
```

Arguments

type_occurrences	Type occurrences matrix
CT	Distinction between C>T at CpG and C>T at other sites, default = FALSE
by	Optional grouping variable
indv_points	Whether to plot the individual samples as points, default = FALSE
error_bars	The type of error bars to plot. * '95 * 'stdev' for standard deviations; * 'SEM' for the standard error of the mean (NOT recommended); * 'none' Do not plot any error bars;
colors	Optional color vector with 7 values
legend	Plot legend, default = TRUE
condensed	More condensed plotting format. Default = F.

Value

Spectrum plot

See Also

read_vcfs_as_granges, mut_type_occurrences

Examples

```
## See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
vcfs <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))


## Load a reference genome.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library(ref_genome, character.only = TRUE)

## Get the type occurrences for all VCF objects.
type_occurrences <- mut_type_occurrences(vcfs, ref_genome)

## Plot the point mutation spectrum over all samples
plot_spectrum(type_occurrences)

## Or with distinction of C>T at CpG sites
plot_spectrum(type_occurrences, CT = TRUE)

## You can also include individual sample points.
plot_spectrum(type_occurrences, CT = TRUE, indv_points = TRUE)

## You can also change the type of error bars
plot_spectrum(type_occurrences, error_bars = "stdev")

## Or plot spectrum per tissue
tissue <- c(
  "colon", "colon", "colon",
  "intestine", "intestine", "intestine",
  "liver", "liver", "liver"
)

plot_spectrum(type_occurrences, by = tissue, CT = TRUE)

## Or plot the spectrum per sample. Error bars are set to 'none', because they can't be plotted.
plot_spectrum(type_occurrences, by = names(vcfs), CT = TRUE, error_bars = "none")

## Plot it in a more condensed manner,
## which is is ideal for publications.
plot_spectrum(type_occurrences,
by = names(vcfs),
CT = TRUE,
error_bars = "none",
condensed = TRUE)

## You can also set custom colors.
my_colors <- c(
  "pink", "orange", "blue", "lightblue",
  "green", "red", "purple"
)
```

```
## And use them in a plot.
plot_spectrum(type_occurrences,
  CT = TRUE,
  legend = TRUE,
  colors = my_colors
)
```

---

plot_spectrum_region *Plot point mutation spectrum per genomic region*

Description

A spectrum similar to the one from 'plot_spectrum()' is plotted. However the spectrum is plotted separately per genomic region. As input it takes a 'type_occurrences' matrix that was calculated per genomic region. To get a 'type_occurrences' matrix per region, first use the 'split_muts_region()' function on a GR or GRangesList. Then use the 'mut_type_occurrences' as you would normally. The by, colors and legend argument work the same as in 'plot_spectrum()'.

Usage

```
plot_spectrum_region(
  type_occurrences,
  by = NA,
  mode = c("relative_sample_feature", "relative_sample", "absolute"),
  indv_points = FALSE,
  error_bars = c("95%_CI", "stdev", "SEM", "none"),
  colors = NULL,
  legend = TRUE,
  condensed = FALSE
)
```

Arguments

type_occurrences	Type occurrences matrix
by	Optional grouping variable
mode	The y-axis plotting mode. * 'relative_sample', the number of variants will be shown divided by the total number of variants in that sample; * 'relative_sample_feature', the number of variants will be shown divided by the total number of variants in that sample and genomic region (Default); * 'absolute' The absolute number of mutations is shown;
indv_points	Whether to plot the individual samples as points, default = FALSE
error_bars	The type of error bars to plot. * '95 * 'stdev' for standard deviations; * 'SEM' for the standard error of the mean (NOT recommended); * 'none' Do not plot any error bars;

colors	Optional color vector with 7 values
legend	Plot legend, default = TRUE
condensed	More condensed plotting format. Default = F.

Details

The y-axis can be plotted with three different modes. With 'relative_sample_feature', the number of variants will be shown divided by the total number of variants in that sample and genomic region. This is generally the most useful, because it allows you to compare the spectra off different regions. When you use 'relative_sample', the number of variants will be shown divided by the total number of variants in that sample. This can be useful when you want to compare the number of mutations between regions. Finally, when you use 'absolute', the absolute mutation numbers are shown. This can be useful when you want to compare the mutation load between different groups of samples.

Value

Spectrum plot by genomic region

See Also

read_vcfs_as_granges, mut_type_occurrences, plot_spectrum, split_muts_region

Other genomic_regions: bin_mutation_density(), lengthen_mut_matrix(), plot_profile_region(), split_muts_region()

Examples

```
## See the 'split_muts_region()' example for how we obtained the
## following data:
grl <- readRDS(system.file("states/grl_split_region.rds",
  package = "MutationalPatterns"
))


## Load a reference genome.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library(ref_genome, character.only = TRUE)


## Get the type occurrences for all VCF objects.
type_occurrences <- mut_type_occurrences(grl, ref_genome)

## Plot the relative point mutation spectrum per genomic region
plot_spectrum_region(type_occurrences)

## Include the individual sample points
plot_spectrum_region(type_occurrences, indv_points = TRUE)

## Plot the relative point mutation spectrum per genomic region,
## but normalize only for the samples
plot_spectrum_region(type_occurrences, mode = "relative_sample")
```

```
## Plot the absolute point mutation spectrum per genomic region
plot_spectrum_region(type_occurrences, mode = "absolute")

## Plot the point mutations spectrum with different error bars
plot_spectrum_region(type_occurrences, error_bars = "stdev")

## Plot the relative point mutation spectrum per sample type and per genomic region
## Determine tissue names
tissue <- c(
  "colon", "colon", "colon",
  "intestine", "intestine", "intestine",
  "liver", "liver", "liver"
)
plot_spectrum_region(type_occurrences, by = tissue)

## Plot the relative point mutation spectrum per individual sample and per genomic region
## Determine sample names
sample_names <- c(
  "colon1", "colon2", "colon3",
  "intestine1", "intestine2", "intestine3",
  "liver1", "liver2", "liver3"
)

plot_spectrum_region(type_occurrences, by = sample_names, error_bars = "none")

## Plot it in a more condensed manner,
## which is is ideal for publications.
plot_spectrum_region(type_occurrences,
by = sample_names,
error_bars = "none",
condensed = TRUE)
```

`plot_strand` *Plot strand per base substitution type*

Description

For each base substitution type and transcriptional strand the total number of mutations and the relative contribution within a group is returned.

Usage

```
plot_strand(strand_bias_df, mode = c("relative", "absolute"), colors = NA)
```

Arguments

`strand_bias_df` data.frame, result from strand_bias function

mode	Either "absolute" for absolute number of mutations, or "relative" for relative contribution, default = "relative"
colors	Optional color vector for plotting with 6 values

Value

Barplot

See Also

mut_matrix_stranded, strand_occurrences, plot_strand_bias

Examples

```
## See the 'mut_matrix_stranded()' example for how we obtained the
## following mutation matrix.
mut_mat_s <- readRDS(system.file("states/mut_mat_s_data.rds",
  package = "MutationalPatterns"
))

## Load a reference genome.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library(ref_genome, character.only = TRUE)

tissue <- c(
  "colon", "colon", "colon",
  "intestine", "intestine", "intestine",
  "liver", "liver", "liver"
)

strand_counts <- strand_occurrences(mut_mat_s, by = tissue)

## Plot the strand in relative mode.
strand_plot <- plot_strand(strand_counts)

## Or absolute mode.
strand_plot <- plot_strand(strand_counts, mode = "absolute")
```

---

plot_strand_bias *Plot strand bias per base substitution type per group*

Description

Plot strand bias per base substitution type per group

Usage

```
plot_strand_bias(strand_bias, colors = NA, sig_type = c("fdr", "p"))
```

Arguments

strand_bias	data.frame, result from strand_bias function
colors	Optional color vector with 6 values for plotting
sig_type	The type of significance to be used. Possible values: * 'fdr' False discovery rate. A type of multiple testing correction.; * 'p' for regular p values.

Value

Barplot

See Also

mut_matrix_stranded, strand_occurrences, strand_bias_test plot_strand

Examples

```
## See the 'mut_matrix_stranded()' example for how we obtained the
## following mutation matrix.
mut_mat_s <- readRDS(system.file("states/mut_mat_s_data.rds",
  package = "MutationalPatterns"
))


tissue <- c(
  "colon", "colon", "colon",
  "intestine", "intestine", "intestine",
  "liver", "liver", "liver"
)

## Perform the strand bias test.
strand_counts <- strand_occurrences(mut_mat_s, by = tissue)
strand_bias <- strand_bias_test(strand_counts)

## Plot the strand bias.
plot_strand_bias(strand_bias)

## Use multiple (max 3) significance cutoffs.
## This will vary the number of significance stars.
strand_bias_multistars <- strand_bias_test(strand_counts,
  p_cutoffs = c(0.05, 0.01, 0.005),
  fdr_cutoffs = c(0.1, 0.05, 0.01)
)
plot_strand_bias(strand_bias_multistars)
```

pool_mut_mat *Pool multiple samples from a mutation matrix together*

Description

The mutation counts of columns (samples) are added up according to the grouping variable.

Usage

```
pool_mut_mat(mut_matrix, grouping)
```

Arguments

mut_matrix	Mutation count matrix (dimensions: x mutation types X n samples)
grouping	Grouping variable

Value

Mutation count matrix (dimensions: x mutation types X n groups)

Examples

```
## See the 'mut_matrix()' example for how we obtained the mutation matrix:
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))
grouping <- c(rep("colon", 3), rep("intestine", 3), rep("liver", 3))
pool_mut_mat(mut_mat, grouping)
```

read_vcfs_as_granges *Read VCF files into a GRangesList*

Description

This function reads Variant Call Format (VCF) files into a GRanges object and combines them in a GRangesList. In addition to loading the files, this function applies the same seqlevel style to the GRanges objects as the reference genome passed in the 'genome' parameter. By default only reads in snv variants.

Usage

```
read_vcfs_as_granges(
  vcf_files,
  sample_names,
  genome,
  group = c("auto+sex", "auto", "sex", "circular", "all", "none"),
  type = c("snv", "indel", "dbs", "mbs", "all"),
  change_seqnames = TRUE,
  predefined_dbs_mbs = FALSE,
  remove_duplicate_variants = TRUE
)
```

Arguments

vcf_files	Character vector of VCF file names
sample_names	Character vector of sample names
genome	BSgenome reference genome object
group	Selector for a seqlevel group. All seqlevels outside of this group will be removed. Possible values: * 'all' for all chromosomes; * 'auto' for autosomal chromosomes; * 'sex' for sex chromosomes; * 'auto+sex' for autosomal + sex chromosomes (default); * 'circular' for circular chromosomes; * 'none' for no filtering, which results in keeping all seqlevels from the VCF file.
type	The mutation type that will be loaded. All other variants will be filtered out. Possible values: * 'snv' (default) * 'indel' * 'dbs' * 'mbs' * 'all' When you use 'all', no filtering or merging of variants is performed.
change_seqnames	Boolean. Whether to change the seqlevelsStyle of the vcf to that of the BSgenome object. (default = TRUE)
predefined_dbs_mbs	Boolean. Whether DBS and MBS variants have been predefined in your vcf. This function by default assumes that DBS and MBS variants are present in the vcf as SNVs, which are positioned next to each other. If your DBS/MBS variants are called separately you should set this argument to TRUE. (default = FALSE)
remove_duplicate_variants	Boolean. Whether duplicate variants are removed. This is based on genomic coordinates and does not take the alternative bases into account. It is generally recommended to keep this on. Turning this off can result in warnings in plot_rainfall. When a duplicate SNV is identified as part of a DBS, only a single DBS, instead of a duplicate DBS will be formed. (default = TRUE)

Value

A GRangesList containing the GRanges obtained from 'vcf_files'

Examples

```
## The example data set consists of three colon samples, three intestine
## samples and three liver samples.  So, to map each file to its appropriate
## sample name, we create a vector containing the sample names:
sample_names <- c(
  "colon1", "colon2", "colon3",
  "intestine1", "intestine2", "intestine3",
  "liver1", "liver2", "liver3"
)

## We assemble a list of files we want to load.  These files match the
## sample names defined above.
vcf_files <- list.files(system.file("extdata",
  package = "MutationalPatterns"
),
pattern = "sample.vcf", full.names = TRUE
)

## Get a reference genome BSgenome object.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library("BSgenome")
library(ref_genome, character.only = TRUE)

## This function loads the files as GRanges objects.
## For backwards compatability reasons it only loads SNVs by default
vcfs <- read_vcfs_as_granges(vcf_files, sample_names, ref_genome)

## To load all variant types use:
vcfs <- read_vcfs_as_granges(vcf_files, sample_names, ref_genome, type = "all")

## Loading only indels can be done like this.

## Select data containing indels.
vcf_fnames <- list.files(system.file("extdata", package = "MutationalPatterns"),
  pattern = "blood.*vcf", full.names = TRUE
)
sample_names <- c("AC", "ACC55", "BCH")

## Read data and select only the indels.
## Other mutation types can be read in the same way.
read_vcfs_as_granges(vcf_fnames, sample_names, ref_genome, type = "indel")
```

---

region_cossim-class *An S4 class to store the results of a regional mutation pattern similarity analysis*

Description

An S4 class to store the results of a regional mutation pattern similarity analysis

Slots

`sim_tb` A tibble containing the calculated similarities of the windows.

`pos_tb` A tibble containing the mutation positions.

`chr_lengths` Vector containing the chromosome lengths.

`window_size` The number of mutations in a window.

`max_window_size_gen` The maximum size of a window before it is removed.

`ref_genome` BSgenome reference genome object

`muts_per_chr` Vector containing the number of mutations per chromosome.

`mean_window_size` The mean length of the genome covered by the windows.

`stepsize` The number of mutations that a window slides in each step.

`extension` The number of bases, that's extracted upstream and downstream of the base substitutions, to create the mutation matrices.

`chromosomes` Vector of chromosome/contig names of the reference genome to be plotted.

`exclude_self_mut_mat` Boolean describing whether the mutations in a window should be subtracted from the global mutation matrix.

`rename_nmf_signatures` *Rename NMF signatures based on previously defined signatures*

Description

This function renames signatures identified with NMF based on previously defined signatures. If a NMF signature has a cosine similarity with a previously defined signature, that is higher than the cutoff, then this NMF signature will get the name of the previously defined signature. If not the NMF signature will receive a letter based name. For example: SBSA. This only changes the names of signatures, not their actual values. This function can be help with identifying whether signatures found with NMF are already known, which can be useful for interpretation. An extracted signature that is not similar to any previously defined signatures, is not proof of a "novel" signature. The extracted signature might be a combination of known signatures, that could not be split by NMF. This can happen when, for example, too few samples were used for the NMF.

Usage

```
rename_nmf_signatures(
  nmf_res,
  signatures,
  cutoff = 0.85,
  base_name = "SBS",
  suffix = "-like"
)
```

Arguments

nmf_res	Named list of mutation matrix, signatures and signature contribution
signatures	A signature matrix
cutoff	Cutoff at which signatures are considered similar. Default: 0.85
base_name	The base part of a letter based signature name. Default: "SBS"
suffix	String. The suffix added to the name of a renamed signature. Default: "-like"

Value

A nmf_res with changed signature names

Examples

```
## Extracting signatures can be computationally intensive, so
## we use pre-computed data generated with the following command:
# nmf_res <- extract_signatures(mut_mat, rank = 2)

nmf_res <- readRDS(system.file("states/nmf_res_data.rds",
  package = "MutationalPatterns"
))

## Get signatures
signatures <- get_known_signatures()

rename_nmf_signatures(nmf_res, signatures)

## You can change or remove the suffix of the renamed signatures.
rename_nmf_signatures(nmf_res, signatures, suffix = "")

## You can change how similar the signatures have to be, before they are considered similar.
rename_nmf_signatures(nmf_res, signatures, cutoff = 0.95)

## You can also change the base_name of the signatures that end up with a letter name.
rename_nmf_signatures(nmf_res, signatures, cutoff = 0.95, base_name = "Signature_")
```

---

show,region_cossim-method

An S4 method to show an instance of the region_cossim class.

Description

An S4 method to show an instance of the region_cossim class.

Usage

```
## S4 method for signature 'region_cossim'
show(object)
```

Arguments

object	A region_cossim object.

`signature_potential_damage_analysis`
Potential damage analysis for the supplied mutational signatures

---

Description

The ratio of possible 'stop gain', 'mismatches', 'synonymous mutations' and 'splice site mutations' is counted per signature. Normalized ratios are also given. These were calculated by dividing the ratios in each signature, by the ratios of a completely "flat" signature. A normalized ratio of 2 for "stop gain" mutations, means that a signature is twice as likely to cause "stop gain" mutations, compared to a completely random "flat" signature. N is the number of possible mutations per context, multiplied by the signature contribution per context, summed over all contexts. For mismatches the blosum62 score is also calculated. A lower score means that the amino acids in the mismatches are more dissimilar. More dissimilar amino acids are more likely to have a detrimental effect. Normalized blosum62 scores are also given. These are calculated by substracting the score of a completely "flat" signature from the base blosum62 scores.

Usage

```
signature_potential_damage_analysis(signatures, contexts, context_mismatches)
```

Arguments

signatures	Matrix containing signatures
contexts	Vector of mutational contexts to use for the analysis.
context_mismatches	A tibble with the ratio of 'stop gain', 'mismatch', 'synonymous' and 'splice site' mutations per mutation context.

Details

The function uses a tibble with the ratio of 'stop gain', 'mismatch', 'synonymous' and 'splice site' mutations per mutation context as its input. For each signature these ratios are linearly combined based on the signature weights. They are then divided by a "flat" signature to get the normalized ratios. The blosum62 scores are also linearly combined based on the signature weights.

Please take into account that this is a relatively basic analysis, that only looks at mutational contexts. It does not take into account that signatures can be influenced by open/closed chromatin, strand biases, hairpins or other epigenetic features. This analysis is meant to give an indication, not a definitive answer, of how damaging a signature might be. Further analyses might be required, especially when using signatures in a clinical context.

Value

A tibble with the ratio of 'stop gain', 'mismatch', 'synonymous' and 'splice site' mutations per signature.

Examples

```
## Get the signatures
signatures <- get_known_signatures()

## See the 'mut_matrix()' example for how we obtained the
## mutation matrix information:
mut_mat <- readRDS(system.file("states/mut_mat_data.rds",
  package = "MutationalPatterns"
))

contexts <- rownames(mut_mat)

## See the 'context_potential_damage_analysis()' example for how we obtained the
## context_mismatches:
context_mismatches <- readRDS(system.file("states/context_mismatches.rds",
  package = "MutationalPatterns"
))

## Determine the potential damage per signature
signature_potential_damage_analysis(signatures, contexts, context_mismatches)
```

---

split_muts_region *Split GRangesList or GRanges based on a list of regions.*

Description

A GRangesList or GRanges object containing variants is split based on a list of regions. This list can be either a GRangesList or a GRanges object. The result is a GRangesList where each element contains the variants of one sample from one region. Variant that are not in any of the provided region are put in a list of 'other'.

Usage

```
split_muts_region(vcf_list, ranges_grl, include_other = TRUE)
```

Arguments

vcf_list	GRangesList or GRanges object
ranges_grl	GRangesList or GRanges object containing regions of interest
include_other	Boolean. Whether or not to include a "Other" region containing mutations that aren't in any other region.

Value

GRangesList

See Also

Other genomic_regions: bin_mutation_density(), lengthen_mut_matrix(), plot_profile_region(), plot_spectrum_region()

Examples

```
## Read in some existing genomic regions.
## See the 'genomic_distribution()' example for how we obtained the
## following data:
CTCF_g <- readRDS(system.file("states/CTCF_g_data.rds",
  package = "MutationalPatterns"
))
promoter_g <- readRDS(system.file("states/promoter_g_data.rds",
  package = "MutationalPatterns"
))
flanking_g <- readRDS(system.file("states/promoter_flanking_g_data.rds",
  package = "MutationalPatterns"
))

## Combine the regions into a single GRangesList
regions <- GRangesList(promoter_g, flanking_g, CTCF_g)

names(regions) <- c("Promoter", "Promoter flanking", "CTCF")

## Read in some variants.
## See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
grl <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

## Split muts based on the supplied regions
split_muts_region(grl, regions)

## Don't include muts outside of the supplied regions
split_muts_region(grl, regions, include_other = FALSE)
```

---

strand_bias_test *Significance test for strand asymmetry*

Description

This function performs a two sided Poisson test for the ratio between mutations on each strand. Multiple testing correction is also performed.

Usage

```
strand_bias_test(strand_occurrences, p_cutoffs = 0.05, fdr_cutoffs = 0.1)
```

Arguments

strand_occurrences	Dataframe with mutation count per strand, result from 'strand_occurrences()'
p_cutoffs	Significance cutoff for the p value. Default: 0.05
fdr_cutoffs	Significance cutoff for the fdr. Default: 0.1

Value

Dataframe with poisson test P value for the ratio between the two strands per group per base substitution type.

See Also

mut_matrix_stranded, strand_occurrences, plot_strand_bias

Examples

```
## See the 'mut_matrix_stranded()' example for how we obtained the
## following mutation matrix.
mut_mat_s <- readRDS(system.file("states/mut_mat_s_data.rds",
  package = "MutationalPatterns"
))

tissue <- c(
  "colon", "colon", "colon",
  "intestine", "intestine", "intestine",
  "liver", "liver", "liver"
)

## Perform the strand bias test.
strand_counts <- strand_occurrences(mut_mat_s, by = tissue)
strand_bias <- strand_bias_test(strand_counts)

## Use different significance cutoffs for the pvalue and fdr
strand_bias_strict <- strand_bias_test(strand_counts,
  p_cutoffs = 0.01, fdr_cutoffs = 0.05
)

## Use multiple (max 3) significance cutoffs.
## This will vary the number of significance stars.
strand_bias_multistars <- strand_bias_test(strand_counts,
  p_cutoffs = c(0.05, 0.01, 0.005),
  fdr_cutoffs = c(0.1, 0.05, 0.01)
)
```

strand_occurrences *Count occurrences per base substitution type and strand*

Description

For each base substitution type and strand the total number of mutations and the relative contribution within a group is returned.

Usage

```
strand_occurrences(mut_mat_s, by = NA)
```

Arguments

mut_mat_s	192 feature mutation count matrix, result from 'mut_matrix_stranded()'
by	Character vector with grouping info, optional

Value

A data.frame with the total number of mutations and relative contribution within group per base substitution type and strand

See Also

mut_matrix_stranded, plot_strand, plot_strand_bias

Examples

```
## See the 'mut_matrix_stranded()' example for how we obtained the
## following mutation matrix.
mut_mat_s <- readRDS(system.file("states/mut_mat_s_data.rds",
  package = "MutationalPatterns"
))

## Load a reference genome.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library(ref_genome, character.only = TRUE)

tissue <- c(
  "colon", "colon", "colon",
  "intestine", "intestine", "intestine",
  "liver", "liver", "liver"
)

strand_counts <- strand_occurrences(mut_mat_s, by = tissue)
```

type_context *Retrieve context of base substitution types*

Description

A function to extract the bases 3' upstream and 5' downstream of the base substitution types.

Usage

```
type_context(vcf, ref_genome, extension = 1)
```

Arguments

vcf	A CollapsedVCF object
ref_genome	Reference genome
extension	The number of bases, that's extracted upstream and downstream of the base substitutions. (Default: 1).

Value

Mutation types and context character vectors in a named list

See Also

read_vcfs_as_granges, mut_context

Examples

```
## See the 'read_vcfs_as_granges()' example for how we obtained the
## following data:
vcfs <- readRDS(system.file("states/read_vcfs_as_granges_output.rds",
  package = "MutationalPatterns"
))

## Load the corresponding reference genome.
ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
library(ref_genome, character.only = TRUE)

## Get type context
type_context <- type_context(vcfs[[1]], ref_genome)

## Get larger type context
type_context_larger <- type_context(vcfs[[1]], ref_genome, extension = 2)
```

Index