Package 'MutationalPatterns'

October 17, 2020

Type Package

Title Comprehensive genome-wide analysis of mutational processes

Description An extensive toolset for the characterization and visualization of a wide range of mutational patterns in base substitution catalogs.

Version 2.0.0

Date 2018-10-19

Author Francis Blokzijl, Roel Janssen, Ruben van Boxtel, Edwin Cuppen

Maintainer Rurika Oka <R.Oka@prinsesmaximacentrum.nl>, Freek Manders <F.M.Manders@prinsesmaximacentrum.nl>

License MIT + file LICENSE

URL https://doi.org/10.1186/s13073-018-0539-0

Imports stats, parallel, S4Vectors, BiocGenerics (>= 0.18.0), VariantAnnotation (>= 1.18.1), reshape2 (>= 1.4.1), plyr (>= 1.8.3), ggplot2 (>= 2.1.0), pracma (>= 1.8.8), SummarizedExperiment (>= 1.2.2), IRanges (>= 2.6.0), GenomeInfoDb (>= 1.12.0), Biostrings (>= 2.40.0), ggdendro (>= 0.1-20), cowplot (>= 0.9.2)

Depends R (>= 3.4.0), GenomicRanges (>= 1.24.0), NMF (>= 0.20.6)

Suggests BSgenome (>= 1.40.0), BSgenome.Hsapiens.1000genomes.hs37d5 (>= 0.99.1), 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), testthat

biocViews Genetics, SomaticMutation

ZipData NA

LazyData false

RoxygenNote 6.0.1

Encoding UTF-8

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

git_branch RELEASE_3_11

git_last_commit ad0aadd

git_last_commit_date 2020-04-27

Date/Publication 2020-10-16

R topics documented:

`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)
```

Arguments

p	Probability of success
n	Number of trials
x	Observed number of successes

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)
```

---

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

```
## You can download mutational signatures from the COSMIC database:
# sp_url = http://cancer.sanger.ac.uk/cancergenome/assets/signatures_probabilities.txt
# cancer_signatures = read.table(sp_url, sep = "\t", header = T)

## We copied the file into our package for your convenience.
filename <- system.file("extdata/signatures_probabilities.txt",
                        package="MutationalPatterns")
cancer_signatures <- read.table(filename, sep = "\t", header = TRUE)
```

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

## Match the order to MutationalPatterns standard of mutation matrix
order = match(row.names(mut_mat), cancer_signatures$Somatic.Mutation.Type)
## Reorder cancer signatures dataframe
cancer_signatures = cancer_signatures[order,]
## Use trinucletiode changes names as row.names
## row.names(cancer_signatures) = cancer_signatures$Somatic.Mutation.Type
## Keep only 96 contributions of the signatures in matrix
cancer_signatures = as.matrix(cancer_signatures[,4:33])
## Rename signatures to number only
colnames(cancer_signatures) = as.character(1:30)

## Hierarchically cluster the cancer signatures based on cosine similarity
hclust_cancer_signatures = cluster_signatures(cancer_signatures)

## Plot dendrogram
plot(hclust_cancer_signatures)

## Save the signature names in the order of the clustering
sig_order = colnames(cancer_signatures)[hclust_cancer_signatures$order]
```

`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`	96 mutation count matrix (dimensions: 96 mutations X n samples)
`mut_matrix2`	96 mutation count matrix (dimensions: 96 mutations 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

```
## You can download mutational signatures from the COSMIC database:
# sp_url = http://cancer.sanger.ac.uk/cancergenome/assets/signatures_probabilities.txt
# cancer_signatures = read.table(sp_url, sep = "\t", header = T)

## We copied the file into our package for your convenience.
filename <- system.file("extdata/signatures_probabilities.txt",
                        package="MutationalPatterns")
cancer_signatures <- read.table(filename, sep = "\t", header = TRUE)

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

## Match the order to MutationalPatterns standard of mutation matrix
order = match(row.names(mut_mat), cancer_signatures$Somatic.Mutation.Type)
## Reorder cancer signatures dataframe
cancer_signatures = cancer_signatures[order,]
## Use trinucletiode changes names as row.names
## row.names(cancer_signatures) = cancer_signatures$Somatic.Mutation.Type
## Keep only 96 contributions of the signatures in matrix
cancer_signatures = as.matrix(cancer_signatures[,4:33])
## Rename signatures to number only
colnames(cancer_signatures) = as.character(1:30)
```

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

`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 = c())
```

Arguments

x	data.frame result from genomic_distribution()
by	Optional grouping variable, e.g. tissue type

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.
distr_test2 <- enrichment_depletion_test(distr)
```

`explained_by_signatures` *This function has been renamed to 'cos_sim_matrix'.*

Description

This function has been renamed to 'cos_sim_matrix'.

Usage

```
explained_by_signatures(mut_matrix, signatures)
```

Arguments

`mut_matrix`	96 mutation count matrix (dimensions: 96 mutations X n samples)
`signatures`	96 mutation count matrix (dimensions: 96 mutations X m samples)

Value

Matrix with pairwise cosine similarities

See Also

`cos_sim_matrix` `mut_matrix`, `fit_to_signatures`, `plot_cosine_heatmap`

Examples

```
## You can download mutational signatures from the COSMIC database:
# sp_url = http://cancer.sanger.ac.uk/cancergenome/assets/signatures_probabilities.txt
# cancer_signatures = read.table(sp_url, sep = "\t", header = T)

## We copied the file into our package for your convenience.
filename <- system.file("extdata/signatures_probabilities.txt",
                        package="MutationalPatterns")
cancer_signatures <- read.table(filename, sep = "\t", header = TRUE)

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

## Match the order to MutationalPatterns standard of mutation matrix
order = match(row.names(mut_mat), cancer_signatures$Somatic.Mutation.Type)
## Reorder cancer signatures dataframe
cancer_signatures = cancer_signatures[order,]
## Use trinucletiode changes names as row.names
## row.names(cancer_signatures) = cancer_signatures$Somatic.Mutation.Type
## Keep only 96 contributions of the signatures in matrix
cancer_signatures = as.matrix(cancer_signatures[,4:33])
## Rename signatures to number only
colnames(cancer_signatures) = as.character(1:30)

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

`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)
```

Arguments

`mut_matrix`	96 mutation count matrix
`rank`	Number of signatures to extract
`nrun`	Number of iterations, default = 200

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 computational intensive.
# nmf_res <- extract_signatures(mut_mat, rank = 2)
```

`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	96 mutation count matrix (dimensions: 96 mutations X n samples)
signatures	Signature matrix (dimensions: 96 mutations X n signatures)

Value

Named list with signature contributions and reconstructed mutation matrix

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"))

## You can download the signatures from the COSMIC website:
# http://cancer.sanger.ac.uk/cancergenome/assets/signatures_probabilities.txt

## We copied the file into our package for your convenience.
filename <- system.file("extdata/signatures_probabilities.txt",
                        package="MutationalPatterns")
cancer_signatures <- read.table(filename, sep = "\t", header = TRUE)

## Match the order to MutationalPatterns standard of mutation matrix
order = match(row.names(mut_mat), cancer_signatures$Somatic.Mutation.Type)
## Reorder cancer signatures dataframe
cancer_signatures = cancer_signatures[order,]
## Use trinucletiode changes names as row.names
## row.names(cancer_signatures) = cancer_signatures$Somatic.Mutation.Type
## Keep only 96 contributions of the signatures in matrix
cancer_signatures = as.matrix(cancer_signatures[,4:33])
## Rename signatures to number only
colnames(cancer_signatures) = as.character(1:30)


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

---

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	A GRangesList or a list with VCF GRanges objects.
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',
#                              'cell_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"))

# open = getBM(attributes = c('chromosome_name', 'chromosome_start',
#                             'chromosome_end', 'feature_type_name'),
#              filters = "regulatory_feature_type_name",
#              values = "Open chromatin",
#              mart = regulatory)
# open_g = reduce(GRanges(open$chromosome_name,
#                 IRanges(open$chromosome_start,
#                         open$chromosome_end)))

open_g <- readRDS(system.file("states/open_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"))

# TF_binding = getBM(attributes = c('chromosome_name', 'chromosome_start',
#                                   'chromosome_end', 'feature_type_name'),
#                    filters = "regulatory_feature_type_name",
#                    values = "TF binding site",
#                    mart = regulatory)
# TF_binding_g = reduce(GRanges(TF_binding$chromosome_name,
#                               IRanges(TF_binding$chromosome_start,
#                               TF_binding$chromosome_end)))

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

regions <- GRangesList(promoter_g, flanking_g, CTCF_g, open_g, TF_binding_g)

names(regions) <- c("Promoter", "Promoter flanking", "CTCF",
                    "Open chromatin", "TF binding")
```

```
# 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)
```

`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 (publically 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

`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]])
```

mutation_context *This function has been removed. Use 'mut_context' instead.*

Description

This function has been removed. Use 'mut_context' instead.

Usage

```
mutation_context(vcf, ref_genome)
```

Arguments

vcf	A GRanges object
ref_genome	The reference genome

Value

Character vector with the context of the base substitutions

See Also

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)

mut_context <- mut_context(vcfs[[1]], ref_genome)
```

mutation_types *This function has been renamed. Use 'mut_type' instead.*

Description

This function has been renamed. Use 'mut_type' instead.

Usage

```
mutation_types(vcf)
```

Arguments

vcf	A GRanges object.

Value

Character vector with base substitution types

See Also

read_vcfs_as_granges mut_type

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_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

Usage

```
mut_context(vcf, ref_genome)
```

Arguments

vcf	A Granges object
ref_genome	Reference genome

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)

mut_context <- mut_context(vcfs[[1]], ref_genome)
```

`mut_matrix` *Make mutation count matrix of 96 trinucleotides*

Description

Make 96 trinucleotide mutation count matrix

Usage

```
mut_matrix(vcf_list, ref_genome)
```

Arguments

`vcf_list`	List of collapsed vcf objects
`ref_genome`	BSGenome reference genome object

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:
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)

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

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")
```

Arguments

vcf_list	List of collapsed vcf objects
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"

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:
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)

## Transcription strand analysis:
## You can obtain the known genes from the UCSC hg19 dataset using
## Bioconductor:
# if (!requireNamespace("BiocManager", quietly=TRUE))
    # install.packages("BiocManager")
# BiocManager::install("TxDb.Hsapiens.UCSC.hg19.knownGene")
# library("TxDb.Hsapiens.UCSC.hg19.knownGene")

## For this example, we preloaded the data for you:
genes_hg19 <- readRDS(system.file("states/genes_hg19.rds",
                        package="MutationalPatterns"))
```

```
mut_mat_s = mut_matrix_stranded(vcfs, ref_genome, genes_hg19,
                                mode = "transcription")

## 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(vcfs, 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 substitions 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:
# if (!requireNamespace("BiocManager", quietly=TRUE))
    # install.packages("BiocManager")
# BiocManager::install("TxDb.Hsapiens.UCSC.hg19.knownGene")
# library("TxDb.Hsapiens.UCSC.hg19.knownGene")

## For this example, we preloaded the data for you:
genes_hg19 <- readRDS(system.file("states/genes_hg19.rds",
                        package="MutationalPatterns"))

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`	A GRangesList
`ref_genome`	Reference genome

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, 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

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"))

## Extract the signatures.
## This is a computationally intensive task, so we load a precomputed
## version instead.
# nmf_res_strand <- extract_signatures(mut_mat_s, rank = 2)
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, ymax = 0.2, condensed = FALSE)
```

Arguments

`mut_matrix`	96 trinucleotide profile matrix
`colors`	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`

Examples

```
## See the 'mut_matrix_stranded()' example for how we obtained the
## mutation matrix with transcriptional strand 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_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,
  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, 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`

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 orignal profile
plot_compare_profiles(mut_mat[,1],
                      nmf_res$reconstructed[,1],
                      profile_names = c("Original", "Reconstructed"))
```

`plot_contribution` *Plot signature contribution barplot*

Description

Plot contribution of signatures

Usage

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

Arguments

`contribution`	Signature contribution matrix
`signatures`	Signature matrix
`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

```
## 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"))

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

## The following are examples of contribution plots.
plot_contribution(nmf_res$contribution,
                  nmf_res$signature,
                  mode = "relative")

plot_contribution(nmf_res$contribution,
                  nmf_res$signature,
                  mode = "absolute")

plot_contribution(nmf_res$contribution,
                  nmf_res$signature,
                  mode = "absolute",
                  index = c(1,2))

plot_contribution(nmf_res$contribution,
                  nmf_res$signature,
                  mode = "absolute",
                  coord_flip = TRUE)
```

`plot_contribution_heatmap` *Plot signature contribution heatmap*

Description

Plot relative contribution of signatures in a heatmap

Usage

```
plot_contribution_heatmap(contribution, sig_order, cluster_samples = TRUE,
  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.
cluster_samples	Hierarchically cluster samples based on eucledian 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

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"))

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

## Define signature order for plotting
sig_order = c("Signature B", "Signature A")


## Contribution heatmap with signatures in defined order
plot_contribution_heatmap(nmf_res$contribution,
                          sig_order = c("Signature B", "Signature A"))

## Contribution heatmap without sample clustering
plot_contribution_heatmap(nmf_res$contribution,
                          sig_order = c("Signature B", "Signature A"),
                          cluster_samples = FALSE, method = "complete")
```

plot_cosine_heatmap *Plot cosine similarity heatmap*

Description

Plot pairwise cosine similarities in a heatmap.

Usage

```
plot_cosine_heatmap(cos_sim_matrix, col_order, cluster_rows = TRUE,
  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.
cluster_rows	Hierarchically cluster rows based on eucledian distance. Default = TRUE.
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, cluster_signatures

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"))

## You can download the signatures from the COSMIC:
# http://cancer.sanger.ac.uk/cancergenome/assets/signatures_probabilities.txt

## We copied the file into our package for your convenience.
filename <- system.file("extdata/signatures_probabilities.txt",
                        package="MutationalPatterns")
cancer_signatures <- read.table(filename, sep = "\t", header = TRUE)

## Match the order to MutationalPatterns standard of mutation matrix
order = match(row.names(mut_mat), cancer_signatures$Somatic.Mutation.Type)
## Reorder cancer signatures dataframe
cancer_signatures = cancer_signatures[order,]
## Use trinucletiode changes names as row.names
```

```
## row.names(cancer_signatures) = cancer_signatures$Somatic.Mutation.Type
## Keep only 96 contributions of the signatures in matrix
cancer_signatures = as.matrix(cancer_signatures[,4:33])
## Rename signatures to number only
colnames(cancer_signatures) = as.character(1:30)

## Calculate the cosine similarity between each signature and each 96 mutational profile
cos_matrix = cos_sim_matrix(mut_mat, cancer_signatures)

## Cluster signatures based on cosine similarity
sig_hclust = cluster_signatures(cancer_signatures)
col_order = colnames(cancer_signatures)[sig_hclust$order]

## Plot the cosine similarity between each signature and each sample with hierarchical
## sample clustering and signatures order based on similarity

plot_cosine_heatmap(cos_matrix, col_order, cluster_rows = TRUE, method = "complete")
```

`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)
```

Arguments

`df`	Dataframe result from enrichment_depletion_test()

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)
distr_test2 = enrichment_depletion_test(distr)

## Plot the enrichment/depletion
plot_enrichment_depletion(distr_test)
plot_enrichment_depletion(distr_test2)
```

`plot_rainfall` *Plot genomic rainfall*

Description

Rainfall plot visualizes the types of mutations and intermutation distance

Usage

```
plot_rainfall(vcf, chromosomes, title = "", colors, cex = 2.5,
  cex_text = 3, ylim = 1e+08)
```

Arguments

`vcf`	CollapsedVCF 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)

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 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)
```

---

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")

plot_signature_strand_bias(nmf_res_strand$signatures)
```

`plot_spectrum` *Plot point mutation spectrum*

Description

Plot point mutation spectrum

Usage

```
plot_spectrum(type_occurrences, CT = FALSE, by, colors, legend = TRUE)
```

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
`colors`	Optional color vector with 7 values
`legend`	Plot legend, default = TRUE

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)

## Or without legend
plot_spectrum(type_occurrences, CT = TRUE, legend = FALSE)

## Or plot spectrum per tissue
tissue <- c("colon", "colon", "colon",
            "intestine", "intestine", "intestine",
            "liver", "liver", "liver")
plot_spectrum(type_occurrences, by = tissue, CT = 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_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 = "relative", colors)
```

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)
```

Arguments

strand_bias	data.frame, result from strand_bias function
colors	Optional color vector with 6 values for plotting

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"))

## 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")

## 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)
```

---

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.

Usage

```
read_vcfs_as_granges(vcf_files, sample_names, genome, group = "auto+sex",
  check_alleles = TRUE)
```

Arguments

vcf_files	Character vector of VCF file names
sample_names	Character vector of sample names
genome	A string matching the name of a BSgenome library corresponding to the reference genome of your VCFs

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.

check_alleles — logical. If TRUE (default) positions with insertions, deletions and/or multiple alternative alleles are excluded from the vcf object, since these positions cannot be analysed with this package. This setting can be set to FALSE to speed up processing time only if the input vcf does not contain any of such positions, as these will cause obscure errors.

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 = ".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
vcfs <- read_vcfs_as_granges(vcf_files, sample_names, ref_genome)
```

---

strand_bias_test *Significance test for strand asymmetry*

Description

This function performs a Poisson test for the ratio between mutations on each strand

Usage

```
strand_bias_test(strand_occurrences)
```

Arguments

strand_occurrences — Dataframe with mutation count per strand, result from strand_occurrences()

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"))

## 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")

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

---

strand_from_vcf *This function has been renamed. Use 'mut_strand' instead.*

Description

This function has been renamed. Use 'mut_strand' instead.

Usage

```
strand_from_vcf(vcf, genes)
```

Arguments

vcf	A GRanges object
genes	The genes

Value

Character vector with transcriptional strand information

See Also

mut_strand

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)
```

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)
```

Arguments

vcf	A CollapsedVCF object
ref_genome	Reference genome

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)

type_context <- type_context(vcfs[[1]], ref_genome)
```

Index