Package ‘mutscan’

February 2, 2026

Title Preprocessing and Analysis of Deep Mutational Scanning Data

Version 1.1.0

Description Provides functionality for processing and statistical analysis of multiplexed assays of variant effect (MAVE) and similar data. The package contains functions covering the full workflow from raw FASTQ files to publication-ready visualizations. A broad range of library designs can be processed with a single, unified interface.

Depends R (>= 4.5.0)

Imports BiocGenerics, S4Vectors, methods, SummarizedExperiment, Rcpp, edgeR (>= 3.42.0), dplyr, Matrix, limma, tidyr, stats, GGally, ggplot2, tidyselect (>= 1.2.0), tibble, rlang, grDevices, csaw, rmarkdown, xfun, DT, ggrepel, IRanges, utils, DelayedArray, tools

Suggests testthat (>= 3.0.0), BiocStyle, knitr, Biostrings, pwalign, plotly, scattermore, BiocManager

SystemRequirements GNU make

biocViews GeneticVariability, GenomicVariation, Preprocessing

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

LinkingTo Rcpp

Config/testthat/edition 3

URL `https://github.com/fmicompbio/mutscan`

BugReports `https://github.com/fmicompbio/mutscan/issues`

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

git_branch devel

git_last_commit 30e3669

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2026-02-01

Author Charlotte Soneson [aut, cre] (ORCID:
<https://orcid.org/0000-0003-3833-2169>),
Michael Stadler [aut] (ORCID: <https://orcid.org/0000-0002-2269-4934>),
Friedrich Miescher Institute for Biomedical Research [cph]

Maintainer Charlotte Soneson <`charlottesoneson@gmail.com`>

Contents

calcNearestStringDist 2
calculateFitnessScore 3
calculateRelativeFC 4
collapseMutantsBySimilarity 6
digestFastqs 8
generateQCReport 15
groupSimilarSequences 17
linkMultipleVariants 18
plotDistributions 20
plotFiltering 21
plotMeanDiff 22
plotPairs 23
plotTotals 25
plotVolcano 26
relabelMutPositions 27
summarizeExperiment 28

Index **30**

`calcNearestStringDist` *Calculate distances to the nearest string*

Description

Given a character vector, calculate the distance for each element to the nearest neighbor amongst all the other elements.

Usage

```
calcNearestStringDist(x, metric = "hamming", nThreads = 1L)
```

Arguments

`x`	A character vector.
`metric`	A character scalar defining the string distance metric. One of `"hamming"` (default), `"hamming_shift"` or `"levenshtein"`.
`nThreads`	numeric(1), number of threads to use for parallel processing.

Value

An integer vector of the same length as `x`.

Examples

```
calcNearestStringDist(c("lazy", "hazy", "crazy"))
calcNearestStringDist(c("lazy", "hazy", "crazy"), metric = "hamming_shift")
calcNearestStringDist(c("lazy", "hazy", "crazy"), metric = "levenshtein")
```

`calculateFitnessScore` *Calculate fitness scores.*

Description

Using sequence counts before and after selection, calculate fitness scores as described by Diss and Lehner (2018).

Usage

```
calculateFitnessScore(
  se,
  pairingCol,
  ODCols,
  comparison,
  WTrows,
  selAssay = "counts"
)
```

Arguments

`se`	SummarizedExperiment object as returned by `summarizeExperiment`.
`pairingCol`	Name of column in `colData(se)` with replicate/pairing information. Samples with the same value in this column will be paired.
`ODCols`	Name(s) of column(s) in `colData(se)` with OD values (numeric), used to normalize for different numbers of cells.
`comparison`	3-element character vector of the form `(column, numerator, denominator)`. `column` is the name of the column in `colData(se)` with experimental conditions. `numerator` and `denominator` define the comparison, e.g. `c("cond", "output", "input")` will look in the `"cond"` column and calculate fitness for the ratio of `"output"` over `"input"` counts.
`WTrows`	Vector of row names that will be used as the reference when calculating fitness scores. If more than one value is provided, the average of the corresponding fitness scores is used as a reference. If NULL, no division by WT scores will be done.
`selAssay`	Assay to select from `se` for the analysis.

Value

A numeric vector with fitness scores.

Author(s)

Michael Stadler and Charlotte Soneson

References

"The genetic landscape of a physical interaction." Diss G and Lehner B. Elife. 2018;7:e32472. doi: 10.7554/eLife.32472.

Examples

```
se <- readRDS(system.file("extdata", "GSE102901_cis_se.rds",
                          package = "mutscan"))
## Check that the wildtype sequence is present in the data
stopifnot("f.0.WT" %in% rownames(se))
## Calculate PPI scores as defined in Diss & Lehner (2018)
ppis <- calculateFitnessScore(
    se = se, pairingCol = "Replicate",
    ODCols = c("OD1", "OD2"),
    comparison = c("Condition", "cis_output", "cis_input"),
    WTrows = "f.0.WT")
## Matrix with PPI scores for each replicate
head(ppis)
```

`calculateRelativeFC` *Calculate logFCs relative to WT using edgeR*

Description

Calculate logFCs and associated p-values for a given comparison, using either limma or the Negative Binomial quasi-likelihood framework of edgeR. The observed counts for the WT variants can be used as offsets in the model.

Usage

```
calculateRelativeFC(
  se,
  design,
  coef = NULL,
  contrast = NULL,
  WTrows = NULL,
  selAssay = "counts",
  pseudocount = 1,
  method = "edgeR",
  normMethod = ifelse(is.null(WTrows), "TMM", "sum")
)
```

Arguments

`se`	SummarizedExperiment object.
`design`	Design matrix. The rows of the design matrix must be in the same order as the columns in `se`.
`coef`	Coefficient(s) to test with edgeR or limma.
`contrast`	Numeric contrast to test with edgeR or limma.
`WTrows`	Vector of row names that will be used as the reference when calculating logFCs and statistics. If more than one value is provided, the sum of the corresponding counts is used to generate offsets. If `NULL`, offsets will be defined as the effective library sizes (using TMM normalization factors).
`selAssay`	Assay to select from `se` for the analysis.
`pseudocount`	Pseudocount to add when calculating log-fold changes.
`method`	Either 'edgeR' or 'limma'. If set to 'limma', voom is used to transform the counts and estimate observation weights before applying limma. In this case, the results also contain the standard errors of the logFCs.
`normMethod`	Character scalar indicating which normalization method should be used to calculate size factors. Should be either `"TMM"` or `"csaw"` when `WTrows` is `NULL`, and `"geomean"` or `"sum"` when `WTrows` is provided.

Value

A `data.frame` with output from the statistical testing framework (edgeR or limma).

Author(s)

Charlotte Soneson, Michael Stadler

Examples

```
library(SummarizedExperiment)
se <- readRDS(system.file("extdata", "GSE102901_cis_se.rds",
                          package = "mutscan"))[1:200, ]
design <- model.matrix(~ Replicate + Condition,
                       data = colData(se))

## Calculate "absolute" log-fold changes with edgeR
res <- calculateRelativeFC(se, design, coef = "Conditioncis_output",
                           method = "edgeR")
head(res)
## Calculate log-fold changes relative to the WT sequence with edgeR
stopifnot("f.0.WT" %in% rownames(se))
res <- calculateRelativeFC(se, design, coef = "Conditioncis_output",
                           method = "edgeR", WTrows = "f.0.WT")
head(res)

## Calculate "absolute" log-fold changes with limma
res <- calculateRelativeFC(se, design, coef = "Conditioncis_output",
                           method = "limma")
```

```
head(res)
## Calculate log-fold changes relative to the WT sequence with limma
stopifnot("f.0.WT" %in% rownames(se))
res <- calculateRelativeFC(se, design, coef = "Conditioncis_output",
                           method = "limma", WTrows = "f.0.WT")
head(res)
```

collapseMutantsBySimilarity

Collapse mutants by similarity

Description

These functions can be used to collapse variants, either by similarity or according to a pre-defined grouping. The functions `collapseMutants` and `collapseMutantsByAA` assume that a grouping variable is available as a column in `rowData(se)` (`collapseMutantsByAA` is a convenience function for the case when this column is "mutantNameAA", and is provided for backwards compatibility). The `collapseMutantsBySimilarity` will generate the grouping variable based on user-provided thresholds on the sequence similarity (defined by the Hamming distance), and subsequently collapse based on the derived grouping.

Usage

```
collapseMutantsBySimilarity(
  se,
  assayName,
  scoreMethod = "rowSum",
  sequenceCol = "sequence",
  collapseMaxDist = 0,
  collapseMinScore = 0,
  collapseMinRatio = 0,
  verbose = TRUE
)

collapseMutantsByAA(se)

collapseMutants(se, nameCol)
```

Arguments

`se`	A `SummarizedExperiment` generated by `summarizeExperiment`
`assayName`	The name of the assay that will be used to calculate a "score" (typically derived from the read counts) for each variant.
`scoreMethod`	Character scalar giving the approach used to calculate ranking scores from the assay defined by `assayName`. Currently, this can be one of `"rowSum"` or `"rowMean"`. All filtering criteria will be applied to these scores.

`sequenceCol`	Character scalar giving the name of the column in `rowData(se)` that contains the nucleotide sequence of the variants.
`collapseMaxDist`	Numeric scalar defining the tolerance for collapsing similar sequences. If the value is in [0, 1), it defines the maximal Hamming distance in terms of a fraction of sequence length: (`round(collapseMaxDist * nchar(sequence))`). A value greater or equal to 1 is rounded and directly used as the maximum allowed Hamming distance. Note that sequences can only be collapsed if they are all of the same length.
`collapseMinScore`	Numeric scalar, indicating the minimum score for the sequence to be considered for collapsing with similar sequences.
`collapseMinRatio`	Numeric scalar. During collapsing of similar sequences, a low-frequency sequence will be collapsed with a higher-frequency sequence only if the ratio between the high-frequency and the low-frequency scores is at least this high. The default value of 0 indicates that no such check is performed.
`verbose`	Logical, whether to print progress messages.
`nameCol`	A character scalar providing the column of `rowData(se)` that contains the amino acid mutant names (that will be the new row names).

Value

A `SummarizedExperiment` where counts have been aggregated by the mutated amino acid(s).

Author(s)

Charlotte Soneson, Michael Stadler

Examples

```
library(SummarizedExperiment)
se <- readRDS(system.file("extdata", "GSE102901_cis_se.rds",
                          package = "mutscan"))[1:200, ]
## The rows of this object correspond to individual codon variants
dim(se)
head(rownames(se))

## Collapse by amino acid
sec <- collapseMutantsByAA(se)
## The rows of the collapsed object correspond to amino acid variants
dim(sec)
head(rownames(sec))
## The mutantName column contains the individual codon variants that were
## collapsed
head(rowData(sec))

## Collapse similar sequences
sec2 <- collapseMutantsBySimilarity(
    se = se, assayName = "counts", scoreMethod = "rowSum",
    sequenceCol = "sequence", collapseMaxDist = 2,
    collapseMinScore = 0, collapseMinRatio = 0)
dim(sec2)
head(rownames(sec2))
head(rowData(sec2))
## collapsed count matrix
assay(sec2, "counts")
```

digestFastqs *Read, filter and digest sequences from fastq file(s).*

Description

Read sequences for one or a pair of fastq files and digest them (extract umis, constant and variable parts, filter, extract mismatch information from constant and count the observed unique variable parts). Alternatively, primer sequences could be specified, in which case the sequence immediately following the primer will be considered the variable sequence.

Usage

```
digestFastqs(
  fastqForward,
  fastqReverse = NULL,
  mergeForwardReverse = FALSE,
  minOverlap = 0,
  maxOverlap = 0,
  minMergedLength = 0,
  maxMergedLength = 0,
  maxFracMismatchOverlap = 1,
  greedyOverlap = TRUE,
  revComplForward = FALSE,
  revComplReverse = FALSE,
  adapterForward = "",
  adapterReverse = "",
  elementsForward = "",
  elementLengthsForward = numeric(0),
  elementsReverse = "",
  elementLengthsReverse = numeric(0),
  primerForward = c(""),
  primerReverse = c(""),
  wildTypeForward = "",
  wildTypeReverse = "",
  constantForward = c(""),
  constantReverse = c(""),
  avePhredMinForward = 20,
  avePhredMinReverse = 20,
  variableNMaxForward = 0,
  variableNMaxReverse = 0,
  umiNMax = 0,
  nbrMutatedCodonsMaxForward = 1,
  nbrMutatedCodonsMaxReverse = 1,
  nbrMutatedBasesMaxForward = -1,
  nbrMutatedBasesMaxReverse = -1,
  forbiddenMutatedCodonsForward = "",
  forbiddenMutatedCodonsReverse = "",
  useTreeWTmatch = FALSE,
  collapseToWTForward = FALSE,
  collapseToWTReverse = FALSE,
  mutatedPhredMinForward = 0,
  mutatedPhredMinReverse = 0,
  mutNameDelimiter = ".",
  constantMaxDistForward = -1,
  constantMaxDistReverse = -1,
  umiCollapseMaxDist = 0,
  filteredReadsFastqForward = "",
  filteredReadsFastqReverse = "",
  maxNReads = -1,
  verbose = FALSE,
  nThreads = 1,
  chunkSize = 1e+05,
  maxReadLength = 1024
)
```

Arguments

`fastqForward`, `fastqReverse`
: Character vectors, paths to gzipped FASTQ files corresponding to forward and reverse reads, respectively. If more than one forward/reverse sequence file is given, they need to be provided in the same order. Note that if multiple fastq files are provided, they are all assumed to correspond to the same sample, and will effectively be concatenated.

`mergeForwardReverse`
: Logical scalar, whether to fuse the forward and reverse variable sequences.

`minOverlap`, `maxOverlap`
: Numeric scalar, the minimal and maximal allowed overlap between the forward and reverse reads when merging. Only used if `mergeForwardReverse` is `TRUE`. If set to 0, only overlaps covering the full length of the shortest of the two reads will be considered.

`minMergedLength`, `maxMergedLength`
: Numeric scalar, the minimal and maximal allowed total length of the merged product (if `mergeForwardReverse` is `TRUE`). If set to 0, any length is allowed.

`maxFracMismatchOverlap`
: Numeric scalar, maximal mismatch rate in the overlap. Only used if `mergeForwardReverse` is `TRUE`.

`greedyOverlap`	Logical scalar. If `TRUE`, the first overlap satisfying `minOverlap`, `maxOverlap`, `minMergedLength`, `maxMergedLength` and `maxFracMismatchOverlap` will be retained. If `FALSE`, all valid overlaps will be scored and the one with the highest score (largest number of matches) will be retained.
`revComplForward`, `revComplReverse`	Logical scalar, whether to reverse complement the forward/reverse variable and constant sequences, respectively.
`adapterForward`, `adapterReverse`	Character scalars, the adapter sequence for forward/reverse reads, respectively. If a forward/reverse read contains the corresponding adapter sequence, the sequence pair will be filtered out. If set to `NULL`, no adapter filtering is performed. The number of filtered read pairs are reported in the return value.
`elementsForward`, `elementsReverse`	Character scalars representing the composition of the forward and reverse reads, respectively. The strings should consist only of the letters S (skip), C (constant), U (umi), P (primer), V (variable), and cover the full extent of the read. Most combinations are allowed (and a given letter can appear multiple times), but there can be at most one occurrence of P. If a given letter is included multiple times, the corresponding sequences will be concatenated in the output.
`elementLengthsForward`, `elementLengthsReverse`	Numeric vectors containing the lengths of each read component from `elementsForward/elementsRevers` respectively. If the length of one element is set to -1, it will be inferred from the other lengths (as the remainder of the read). At most one number (or one number on each side of the primer P) can be set to -1. The indicated length of the primer is not used (instead it's inferred from the provided primer sequence) and can also be set to -1.
`primerForward`, `primerReverse`	Character vectors, representing the primer sequence(s) for forward/reverse reads, respectively. Only read pairs that contain perfect matches to both the forward and reverse primers (if given) will be retained. Multiple primers can be specified - they will be considered in order and the first match will be used.
`wildTypeForward`, `wildTypeReverse`	Character scalars or named character vectors, the wild type sequence for the forward and reverse variable region. If given as a single string, the reference sequence will be named 'f' (for forward) or 'r' (for reverse).
`constantForward`, `constantReverse`	Character vectors giving, the expected constant forward and reverse sequences. If more than one sequence is provided, they must all have the same length.
`avePhredMinForward`, `avePhredMinReverse`	Numeric scalar, the minimum average Phred score in the variable region for a read to be retained. If a read pair contains both forward and reverse variable regions, the minimum average Phred score has to be achieved in both for a read pair to be retained.
`variableNMaxForward`, `variableNMaxReverse`	Numeric scalar, the maximum number of Ns allowed in the variable region for a read to be retained.

`umiNMax`	Numeric scalar, the maximum number of Ns allowed in the UMI for a read to be retained.
`nbrMutatedCodonsMaxForward`, `nbrMutatedCodonsMaxReverse`	Numeric scalar, the maximum number of mutated codons that are allowed. Note that for the forward and reverse sequence, respectively, exactly one of `nbrMutatedCodonsMax` and `nbrMutatedBasesMax` must be -1, and the other must be a non-negative number. The one that is not -1 will be used to filter and name the identified mutants.
`nbrMutatedBasesMaxForward`, `nbrMutatedBasesMaxReverse`	Numeric scalar, the maximum number of mutated bases that are allowed. Note that for the forward and reverse sequence, respectively, exactly one of `nbrMutatedCodonsMax` and `nbrMutatedBasesMax` must be -1, and the other must be a non-negative number. The one that is not -1 will be used to filter and name the identified mutants.
`forbiddenMutatedCodonsForward`, `forbiddenMutatedCodonsReverse`	Character vector of codons (can contain ambiguous IUPAC characters, see `IUPAC_CODE_MAP`). If a read pair contains a mutated codon matching this pattern, it will be filtered out.
`useTreeWTmatch`	Logical scalar. Should a tree-based matching to wild type sequences be used if possible? If the number of allowed mismatches is small, and the number of wild type sequences is large, this is typically faster.
`collapseToWTForward`, `collapseToWTReverse`	Logical scalar, indicating whether to just represent the observed variable sequence by the closest wildtype sequence rather than retaining the information about the mutations.
`mutatedPhredMinForward`, `mutatedPhredMinReverse`	Numeric scalar, the minimum Phred score of a mutated base for the read to be retained. If any mutated base has a Phred score lower than `mutatedPhredMin`, the read (pair) will be discarded.
`mutNameDelimiter`	Character scalar, the delimiter used in the naming of mutants. Generally, mutants will be named as XX{.}YY{.}NNN, where XX is the closest provided reference sequence, YY is the mutated base or codon number (depending on whether `nbrMutatedBases*` or `nbrMutatedCodons*` is specified), and NNN is the mutated base or codon. Here, {.} is the provided `mutNameDelimiter`. The delimiter must be a single character (not "_"), and can not appear in any of the provided reference sequence names.
`constantMaxDistForward`, `constantMaxDistReverse`	Numeric scalars, the maximum allowed Hamming distance between the extracted and expected constant sequence. If multiple constant sequences are provided, the most similar one is used. Reads with a larger distance to the expected constant sequence are discarded. If set to -1, no filtering is done.
`umiCollapseMaxDist`	Numeric scalar defining the tolerances for collapsing similar UMI sequences. If the value is in [0, 1), it defines the maximal Hamming distance in terms of a fraction of sequence length: `(round(umiCollapseMaxDist * nchar(umiSeq)))`. A value greater or equal to 1 is rounded and directly used as the maximum allowed Hamming distance.

`filteredReadsFastqForward`, `filteredReadsFastqReverse`	Character scalars, the names of a (pair of) FASTQ file(s) where filtered-out reads will be written. The name(s) should end in .gz (the output will always be compressed). If empty, filtered reads will not be written to a file.
`maxNReads`	Integer scalar, the maximum number of reads to process. The first `maxNReads` read (pairs) in the FASTQ file(s) will be used. If set to -1, all reads in the FASTQ file(s) will be processed.
`verbose`	Logical scalar, whether to print out progress messages.
`nThreads`	Numeric scalar, the number of threads to use for parallel processing.
`chunkSize`	Numeric scalar, the number of read (pairs) to keep in memory for parallel processing. Reduce from the default value if you run out of memory.
`maxReadLength`	Numeric scalar, the maximum allowed read length. Longer read lengths lead to higher memory allocation, and may require the `chunkSize` to be decreased.

Details

The processing of a read pair goes as follows:

1. Search for perfect matches to forward/reverse adapter sequences, filter out the read pair if a match is found in either the forward or reverse read.
2. If primer sequences are provided, search for perfect matches, and filter out the read pair if not all provided primer sequences can be found.
3. Extract the UMI, constant and variable sequence from forward and reverse reads, based on the definition of the respective read composition.
4. If requested, collapse forward and reverse variable regions by retaining, for each position, the base with the highest reported base quality.
5. Filter out the read (pair) if the average quality in the variable region is below `avePhredMinForward/avePhredMinRever` in either the forward or reverse read (or the merged read).
6. Filter out the read (pair) if the number of Ns in the variable region exceeds `variableNMaxForward/variableNMaxRever`
7. Filter out the read (pair) if the number of Ns in the combined forward and reverse UMI sequence exceeds `umiNMax`
8. If one or more wild type sequences (for the variable region) are provided, find the mismatches between the (forward/reverse) variable region and the provided wild type sequence (if more than one wild type sequence is provided, first find the one that is closest to the read).
9. Filter out the read (pair) if any mutated base has a quality below `mutatedPhredMinForward/mutatedPhredMinReverse`
10. Filter out the read (pair) if the number of mutated codons exceeds `nbrMutatedCodonsMaxForward/nbrMutatedCodonsM`
11. Filter out the read (pair) if any of the mutated codons match any of the codons encoded by `forbiddenMutatedCodonsForward/forbiddenMutatedCodonsReverse`.
12. Assign a 'mutation name' to the read (pair). This name is a combination of parts of the form XX{.}YY{.}NNN, where XX is the name of the most similar reference sequence, YY is the mutated codon number, and NNN is the mutated codon. {.} is a delimiter, specified via `mutNameDelimiter`. If no wildtype sequences are provided, the variable sequence will be used as the mutation name'.

Based on the retained reads following this filtering process, count the number of reads, and the number of unique UMIs, for each variable sequence (or pair of variable sequences).

Value

A list with four entries:

summaryTable A `data.frame` that contains, for each observed mutation combination, the corresponding variable region sequences (or pair of sequences), the number of observed such sequences, and the number of unique UMIs observed for the sequence. It also has additional columns: 'maxNbrReads' contains the number of reads for the most frequent observed sequence represented by the feature (only relevant if similar variable regions are collapsed). 'nbrMutBases', 'nbrMutCodons' and 'nbrMutAAs' give the number of mutated bases, codons or amino acids in each variant. Alternative variant names based on base, codon or amino acid sequence are provided in columns mutantNameBase', 'mutantNameCodon', 'mutantNameAA'. In addition, mutantNameBaseHGVS' and 'mutantNameAAHGVS' give base- and amino acid-based names following the HGVS nomenclature (https://varnomen.hgvs.org/). Please note that the provided reference sequence names are used for the HGVS sequence identifiers. It is up to the user to use appropriately named reference sequences in order to obtain valid HGVS variant names.

filterSummary A `data.frame` that contains the number of input reads, the number of reads filtered out in the processing, and the number of retained reads. The filters are named according to the convention "fxx_filter", where "xx" indicates the order in which the filters were applied, and "filter" indicates the type of filter. Note that filters are applied successively, and the reads filtered out in one step are not considered for successive filtering steps.

errorStatistics A `data.frame` that contains, for each Phred quality score between 0 and 99, the number of bases in the extracted constant sequences with that quality score that match/mismatch with the provided reference constant sequence.

parameters A `list` with all parameter settings that were used in the processing. Also contains the version of the package and the time of processing.

Examples

```
## See the vignette for complete worked-out examples for different types of
## data sets

## ---------------------------------------------------------------------- ##
## Process a single-end data set, assume that the full read represents
## the variable region
out <- digestFastqs(
    fastqForward = system.file("extdata", "cisInput_1.fastq.gz",
                               package = "mutscan"),
    elementsForward = "V", elementLengthsForward = -1
)
## Table with read counts and mutant information
head(out$summaryTable)
## Filter summary
out$filterSummary

## ---------------------------------------------------------------------- ##
## Process a single-end data set, specify the read as a combination of
## UMI, constant region and variable region (skip the first base)
out <- digestFastqs(
    fastqForward = system.file("extdata", "cisInput_1.fastq.gz",
                               package = "mutscan"),
    elementsForward = "SUCV", elementLengthsForward = c(1, 10, 18, 96),
    constantForward = "AACCGGAGGAGGGAGCTG"
)
## Table with read counts and mutant information
head(out$summaryTable)
## Filter summary
out$filterSummary
## Error statistics
out$errorStatistics

## ---------------------------------------------------------------------- ##
## Process a single-end data set, specify the read as a combination of
## UMI, constant region and variable region (skip the first base), provide
## the wild type sequence to compare the variable region to and limit the
## number of allowed mutated codons to 1
out <- digestFastqs(
    fastqForward = system.file("extdata", "cisInput_1.fastq.gz",
                               package = "mutscan"),
    elementsForward = "SUCV", elementLengthsForward = c(1, 10, 18, 96),
    constantForward = "AACCGGAGGAGGGAGCTG",
    wildTypeForward = c(FOS = paste0(
        "ACTGATACACTCCAAGCGGAGACAGACCAACTAGAAGATGAGAAGTC",
        "TGCTTTGCAGACCGAGATTGCCAACCTGCTGAAGGAGAAGGAAAAACTA")),
    nbrMutatedCodonsMaxForward = 1
)
## Table with read counts and mutant information
head(out$summaryTable)
## Filter summary
out$filterSummary
## Error statistics
out$errorStatistics

## ---------------------------------------------------------------------- ##
## Process a paired-end data set where both the forward and reverse reads
## contain the same variable region and thus should be merged to generate
## the final variable sequence, specify the reads as a combination of
## UMI, constant region and variable region (skip the first and/or last
## base), provide the wild type sequence to compare the variable region to
## and limit the number of allowed mutated codons to 1
out <- digestFastqs(
    fastqForward = system.file("extdata", "cisInput_1.fastq.gz",
                               package = "mutscan"),
    fastqReverse = system.file("extdata", "cisInput_2.fastq.gz",
                               package = "mutscan"),
    mergeForwardReverse = TRUE,
    revComplForward = FALSE, revComplReverse = TRUE,
    elementsForward = "SUCV", elementLengthsForward = c(1, 10, 18, 96),
    elementsReverse = "SUCVS", elementLengthsReverse = c(1, 7, 17, 96, -1),
    constantForward = "AACCGGAGGAGGGAGCTG",
    constantReverse = "GAGTTCATCCTGGCAGC",
    wildTypeForward = c(FOS = paste0(
        "ACTGATACACTCCAAGCGGAGACAGACCAACTAGAAGATGAGAAGTC",
        "TGCTTTGCAGACCGAGATTGCCAACCTGCTGAAGGAGAAGGAAAAACTA")),
    nbrMutatedCodonsMaxForward = 1
)
## Table with read counts and mutant information
head(out$summaryTable)
## Filter summary
out$filterSummary
## Error statistics
out$errorStatistics

## ---------------------------------------------------------------------- ##
## Process a paired-end data set where the forward and reverse reads
## contain variable regions corresponding to different proteins, and thus
## should not be merged, specify the reads as a combination of
## UMI, constant region and variable region (skip the first base), provide
## the wild type sequence to compare the variable region to and limit the
## number of allowed mutated codons to 1
out <- digestFastqs(
    fastqForward = system.file("extdata", "transInput_1.fastq.gz",
                               package = "mutscan"),
    fastqReverse = system.file("extdata", "transInput_2.fastq.gz",
                               package = "mutscan"),
    mergeForwardReverse = FALSE,
    elementsForward = "SUCV", elementLengthsForward = c(1, 10, 18, 96),
    elementsReverse = "SUCV", elementLengthsReverse = c(1, 8, 20, 96),
    constantForward = "AACCGGAGGAGGGAGCTG",
    constantReverse = "GAAAAAGGAAGCTGGAGAGA",
    wildTypeForward = c(FOS = paste0(
        "ACTGATACACTCCAAGCGGAGACAGACCAACTAGAAGATGAGAAGTC",
        "TGCTTTGCAGACCGAGATTGCCAACCTGCTGAAGGAGAAGGAAAAACTA")),
    wildTypeReverse = c(JUN = paste0(
        "ATCGCCCGGCTGGAGGAAAAAGTGAAAACCTTGAAAGCTCAGAACTC",
        "GGAGCTGGCGTCCACGGCCAACATGCTCAGGGAACAGGTGGCACAGCTT")),
    nbrMutatedCodonsMaxForward = 1,
    nbrMutatedCodonsMaxReverse = 1
)
## Table with read counts and mutant information
head(out$summaryTable)
## Filter summary
out$filterSummary
## Error statistics
out$errorStatistics
```

`generateQCReport` *Generate QC report*

Description

Generate QC report

Usage

```
generateQCReport(
  se,
  outFile,
  reportTitle = "mutscan QC report",
  forceOverwrite = FALSE,
  ...
)
```

Arguments

`se`	A `SummarizedExperiment` object, typically generated with `summarizeExperiment()`.
`outFile`	Character string providing the name of the output file. Should have the extension `.html`.
`reportTitle`	Character string specifying the title of the QC report.
`forceOverwrite`	Logical scalar, indicating whether an existing file with the same name as `outFile` should be overwritten.
`...`	Additional parameters to be forwarded to `render`, for example `quiet = TRUE`.

Value

Invisibly, the path to the generated html file.

Author(s)

Charlotte Soneson

See Also

`render` used to render the html output file.

Examples

```
## Load SummarizedExperiment object
se <- readRDS(system.file("extdata", "GSE102901_cis_se.rds",
                          package = "mutscan"))
## Define output file
outfile <- tempfile(fileext = ".html")

## Generate QC report
generateQCReport(se, outfile)
```

`groupSimilarSequences` *Create a conversion table for collapsing similar sequences*

Description

Create a conversion table for collapsing similar sequences

Usage

```
groupSimilarSequences(
  seqs,
  scores,
  collapseMaxDist = 0,
  collapseMinScore = 0,
  collapseMinRatio = 0,
  verbose = FALSE
)
```

Arguments

`seqs`	Character vector with nucleotide sequences (or pairs of sequences concatenated with "_") to be collapsed. The sequences must all be of the same length.
`scores`	Numeric vector of "scores" for the sequences. Typically the total read/UMI count. A higher score will be preferred when deciding which sequence to use as the representative for a group of collapsed sequences.
`collapseMaxDist`	Numeric scalar defining the tolerance for collapsing similar sequences. If the value is in [0, 1), it defines the maximal Hamming distance in terms of a fraction of sequence length: (`round(collapseMaxDist * nchar(sequence))`). A value greater or equal to 1 is rounded and directly used as the maximum allowed Hamming distance. Note that sequences can only be collapsed if they are all of the same length. The default value is 0.
`collapseMinScore`	Numeric scalar, indicating the minimum score required for a sequence to be considered as a representative for a group of similar sequences (i.e., to allow other sequences to be collapsed into it). The default value is 0.
`collapseMinRatio`	Numeric scalar. During collapsing of similar sequences, a low-frequency sequence will be collapsed with a higher-frequency sequence only if the ratio between the high-frequency and the low-frequency scores is at least this high. A value of 0 indicates that no such check is performed.
`verbose`	Logical scalar, whether to print progress messages.

Value

A data.frame with two columns, containing the input sequences and the representatives for the groups resulting from grouping similar sequences, respectively.

Author(s)

Michael Stadler, Charlotte Soneson

Examples

```
seqs <- c("AACGTAGCA", "ACCGTAGCA", "AACGGAGCA", "ATCGGAGCA", "TGAGGCATA")
scores <- c(5, 1, 3, 1, 8)
groupSimilarSequences(seqs = seqs, scores = scores,
                      collapseMaxDist = 1, collapseMinScore = 0,
                      collapseMinRatio = 0, verbose = FALSE)
```

`linkMultipleVariants` *Process an experiment with multiple variable sequences*

Description

This function enables the processing of data sets with multiple variable sequences, which should potentially be handled in different ways. For example, a barcode association experiment with two variable sequences (the barcode and the biological variant) that need to be processed differently, e.g. in terms of matching to wildtype sequences or collapsing of similar sequences. In contrast, while `digestFastqs` allow the specification of multiple variable sequences (within each of the forward and reverse reads), they will be concatenated and processed as a single unit.

Usage

```
linkMultipleVariants(combinedDigestParams = list(), ...)
```

Arguments

`combinedDigestParams`	A named list of arguments to `digestFastqs` for the combined ("naive") run.
`...`	Additional arguments providing arguments to `digestFastqs` for the separate runs (processing each variable sequence in turn). Each argument must be a named list of arguments to `digestFastqs`. In addition, arguments `collapseMaxDist`, `collapseMinScore` and `collapseMinRatio` can be specified, and will be passed on to `collapseMutantsBySimilarity`.

Details

linkMultipleVariants will process the input in the following way:

- First, run `digestFastqs` with the parameters provided in `combinedDigestParams`. Typically, this will be a "naive" counting run, where the frequencies of all observed variants are tabulated. The variable sequences within the forward and reverse reads, respectively, will be processed as a single sequence.

- Next, run `digestFastqs` with each of the additional parameter sets provided (`...`). Each of these should correspond to a single variable sequence from the combined run (i.e., if there are two Vs in the element specifications in the combined run, there should be two additional parameter sets provided, each corresponding to the processing of one variable sequence part). It is assumed that the order of the additional arguments correspond to the order of the variable sequences in the combined run, in such a way that if the variable sequences extracted in each of the separate runs are concatenated in the order that the parameter sets are provided to `linkMultipleVariants`, they will form the variable sequence extracted in the combined run.
- The result of each of the separate runs is a 'conversion table', containing the final set of identified sequence variants as well as all individual sequences corresponding to each of them. This is then combined with the count table from the combined, "naive" run in order to create an aggregated count table. More precisely, each sequence in the combined run is split into the constituent variable sequences, and each variable sequence is then matched to the output from the right separate run, from which the final feature ID (mutant name, or collapsed sequence) will be extracted and used to replace the original sequence in the combined count table. Once all the matches are done, rows with NAs (where no match could be found in the separate run) are removed and the counts are aggregated across all identical combinations of variable sequences.

In order to define the `elementsForward` and `elementsReverse` arguments for the separate runs, a strategy that often works is to simply copy the arguments from the combined run, and successively replace all but one of the 'V's by 'S'. This will effectively process one variable sequence at the time, while keeping all other elements of the reads consistent (since this can affect e.g. filtering criteria). Note that to process individual variable sequences in the reverse read, you also need to swap the 'forward' and 'reverse' specifications (since `digestFastqs` requires a forward read).

Value

A list with the following elements:

- countAggregated - a `tibble` with columns corresponding to each of the variable sequences, and a column with the total observed read count for the combination.
- convSeparate - a list of conversion tables from the respective separate runs.
- outCombined - the `digestFastqs` output for the combined run.

Author(s)

Charlotte Soneson, Michael Stadler

Examples

```
fqFile <- system.file("extdata", "cisInput_1.fastq.gz",
                      package = "mutscan")
out <- linkMultipleVariants(
    combinedDigestParams = list(fastqForward = fqFile,
                                elementsForward = "SVCV",
                                elementLengthsForward = c(1, 10, 18, 96)),
    # the first variable sequence is the UMI
    umi = list(fastqForward = fqFile, elementsForward = "SVCS",
               elementLengthsForward = c(1, 10, 18, 96)),
    # the second variable sequence is the amplicon variant
    var = list(fastqForward = fqFile, elementsForward = "SSCV",
               elementLengthsForward = c(1, 10, 18, 96),
               collapseMaxDist = 3, collapseMinScore = 1)
)
# conversion tables
lapply(out$convSeparate, head)
# aggregated count table
head(out$countAggregated)
```

---

plotDistributions *Plot distribution of observed values*

Description

Plot distribution of observed values

Usage

```
plotDistributions(
  se,
  selAssay = "counts",
  groupBy = NULL,
  plotType = "density",
  facet = FALSE,
  pseudocount = 0
)
```

Arguments

`se`	A `SummarizedExperiment` object, typically generated by `summarizeExperiment()`.
`selAssay`	Character scalar specifying the assay in `se` to use for the plotting.
`groupBy`	Character scalar specifying a column from `colData(se)` to use for coloring or stratifying the plots.
`plotType`	Character scalar specifying the type of plot to construct. Either `'density'`, `'histogram'` or `'knee'`.
`facet`	Logical scalar, indicating whether or not to facet the plot by the values specified in the `groupBy` column.
`pseudocount`	Numeric scalar, representing the number to add to the observed values in the `selAssay` assay before plotting.

Value

A ggplot object.

Author(s)

Charlotte Soneson

Examples

```
se <- readRDS(system.file("extdata", "GSE102901_cis_se.rds",
                          package = "mutscan"))[1:200, ]
plotDistributions(se)
```

`plotFiltering` *Visualize the filtering procedure*

Description

Display the number (or fraction) of reads remaining after each step of the internal `mutscan` filtering.

Usage

```
plotFiltering(
  se,
  valueType = "reads",
  onlyActiveFilters = TRUE,
  displayNumbers = TRUE,
  numberSize = 4,
  plotType = "remaining",
  facetBy = "sample"
)
```

Arguments

`se`	A `SummarizedExperiment` object, e.g. from `summarizeExperiment`.
`valueType`	Either "reads" or "fractions", indicating whether to plot the number of reads, or the fraction of the total number of reads, that are retained after/filtered out in each filtering step.
`onlyActiveFilters`	Logical scalar, whether to only include the active filters (i.e., where any read was filtered out in any of the samples). Defaults to `TRUE`.
`displayNumbers`	Logical scalar, indicating whether to display the number (or fraction) of reads retained at every filtering step.
`numberSize`	Numeric scalar, indicating the size of the displayed numbers (if `displayNumbers` is `TRUE`).
`plotType`	Character scalar, indicating what to show in the plot. Either `"remaining"` or `"filtered"`.
`facetBy`	Character scalar, indicating the variable by which the plots should be facetted. Either `"sample"` or `"step"`.

Details

The function assumes that the number of reads filtered out in each step are provided as columns of `colData(se)`, with column names of the form `f[0-9]_filteringreason`, and that all filtering columns occur between the columns named `nbrTotal` and `nbrRetained`.

Value

A `ggplot` object.

Author(s)

Charlotte Soneson

Examples

```
se <- readRDS(system.file("extdata", "GSE102901_cis_se.rds",
                          package = "mutscan"))[1:200, ]
plotFiltering(se)
```

`plotMeanDiff` *Construct an MA (mean-difference) plot*

Description

Construct an MA (mean-difference) plot

Usage

```
plotMeanDiff(
  res,
  meanCol = NULL,
  logFCCol = NULL,
  pvalCol = NULL,
  padjCol = NULL,
  padjThreshold = 0.05,
  pointSize = "small",
  interactivePlot = FALSE,
  nTopToLabel = 0
)
```

Arguments

`res`	`data.frame` (typically output from `calculateRelativeFC()`) with columns corresponding to the average abundance (`logCPM` or `AveExpr`), log-fold change (`logFC`) and significance (`FDR` or `adj.P.Val`).

`meanCol`, `logFCCol`, `pvalCol`, `padjCol`	Character scalars indicating the columns from `res` that will be used to represent the mean value (x-axis), logFC (y-axis), nominal p-value (used to find the top features to label) and adjusted p-value (used for coloring). If `NULL` (default), pre-specified values will be used depending on the available columns (`"logCPM"` or `"AveExpr"`, `"logFC"`, `"PValue"` or `"P.Value"`, and `"FDR"` or `"adj.P.Val"`, respectively).
`padjThreshold`	Numeric scalar indicating the adjusted p-value threshold to use for coloring the points. All features with adjusted p-value below the treshold will be shown in red.
`pointSize`	Either `"small"` or `"large"`, indicating which of the two available plot styles that will be used.
`interactivePlot`	Logical scalar, indicating whether an interactive plot should be returned, in which one can hover over the individual points and obtain further information.
`nTopToLabel`	Numeric scalar, indicating the number of points that should be labeled in the plot. The points will be ranked by the `pvalCol` column, and the top `nTopToLabel` values will be labeled by the corresponding row names. Only used if `interactivePlot` is `FALSE`.

Value

If `interactivePlot` is `TRUE`, a `plotly` object. If `interactivePlot` is `FALSE`, a `ggplot2` object.

Author(s)

Charlotte Soneson

Examples

```
library(SummarizedExperiment)
se <- readRDS(system.file("extdata", "GSE102901_cis_se.rds",
                          package = "mutscan"))[1:200, ]
design <- model.matrix(~ Replicate + Condition,
                       data = colData(se))
res <- calculateRelativeFC(se, design, coef = "Conditioncis_output")
plotMeanDiff(res, pointSize = "large", nTopToLabel = 3)
```

`plotPairs` *Make pairs plot of selected assay from a SummarizedExperiment object*

Description

Construct a pairs plot of all columns of a given assay. The lower-triangular panels display the scatter plots, the upper-triangular ones print out the (Pearson or Spearman) correlations, and the diagonal panels show histograms of the respective columns.

Usage

```
plotPairs(
  se,
  selAssay = "counts",
  doLog = TRUE,
  pseudocount = 1,
  corMethod = "pearson",
  histBreaks = 40,
  pointsType = "points",
  corSizeMult = 5,
  corSizeAdd = 2,
  pointSize = 0.1,
  pointAlpha = 0.3,
  colorByCorrelation = TRUE,
  corrColorRange = NULL,
  addIdentityLine = FALSE
)
```

Arguments

`se`	A SummarizedExperiment object, e.g. the output of `summarizeExperiment`
`selAssay`	Character scalar, the assay to use as the basis for the pairs plot.
`doLog`	Logical scalar, whether or not to log-transform the values before plotting.
`pseudocount`	Numeric scalar, the pseudocount to add to the values before log-transforming (if `doLog` is `TRUE`).
`corMethod`	Either "pearson" or "spearman", the type of correlation to calculate.
`histBreaks`	Numeric scalar, the number of breaks in the histograms to put in the diagonal panels.
`pointsType`	Either "points", "smoothscatter", "scattermore" or "scattermost" (the latter two require the "scattermore" package to be installed), determining the type of plots that will be made.
`corSizeMult, corSizeAdd`	Numeric scalars determining how the absolute correlation value is transformed into a font size. The transformation is corSizeMult * abs(corr) + corSizeAdd.
`pointSize, pointAlpha`	Numeric scalars determining the size and opacity of points in the plot.
`colorByCorrelation`	Logical scalar, indicating whether the correlation panels should be colored according to the correlation value.
`corrColorRange`	Numeric vector of length 2, providing the lower and upper limits of the color scale when coloring by correlation. Both values should be positive; the same range is used for negative correlations. If `NULL` (the default), the range is inferred from the data.
`addIdentityLine`	Logical scalar, indicating whether the identity line should be added (only used if `pointsType = "points"`).

Value

A `ggplot` object.

Author(s)

Charlotte Soneson

Examples

```
se <- readRDS(system.file("extdata", "GSE102901_cis_se.rds",
                          package = "mutscan"))[1:200, ]
plotPairs(se)
```

---

plotTotals *Plot the column totals of a selected assay*

Description

Plot the column totals of a selected assay

Usage

```
plotTotals(se, selAssay = "counts", groupBy = NULL)
```

Arguments

`se`	A `SummarizedExperiment` object, typically generated by `summarizeExperiment()`.
`selAssay`	Character scalar specifying the assay in `se` to use for the plotting.
`groupBy`	Character scalar indicating a column in `rowData(se)` to group the features by before calculating the column sums.

Value

A ggplot object.

Author(s)

Charlotte Soneson

Examples

```
se <- readRDS(system.file("extdata", "GSE102901_cis_se.rds",
                          package = "mutscan"))[1:200, ]
plotTotals(se)
```

`plotVolcano` *Construct a volcano plot*

Description

Construct a volcano plot

Usage

```
plotVolcano(
  res,
  logFCCol = NULL,
  pvalCol = NULL,
  padjCol = NULL,
  padjThreshold = 0.05,
  pointSize = "small",
  interactivePlot = FALSE,
  nTopToLabel = 0
)
```

Arguments

`res`	`data.frame` (typically output from `calculateRelativeFC()`) with columns corresponding to the log-fold change (`logFC`), p-value (`PValue` or `P.Value`) and significance (`FDR` or `adj.P.Val`).
`logFCCol`, `pvalCol`, `padjCol`	Character scalars indicating the columns from `res` that will be used to represent the logFC (x-axis), p-value (y-axis) and adjusted p-value (used for coloring). If `NULL` (default), pre-specified values will be used depending on the available columns (`"logFC"`, `"PValue"` or `"P.Value"`, and `"FDR"` or `"adj.P.Val"`, respectively).
`padjThreshold`	Numeric scalar indicating the adjusted p-value threshold to use for coloring the points. All features with adjusted p-value below the treshold will be shown in red.
`pointSize`	Either `"small"` or `"large"`, indicating which of the two available plot styles that will be used.
`interactivePlot`	Logical scalar, indicating whether an interactive plot should be returned, in which one can hover over the individual points and obtain further information.
`nTopToLabel`	Numeric scalar, indicating the number of points that should be labeled in the plot. The points will be ranked by the `pvalCol` column, and the top `nTopToLabel` values will be labeled by the corresponding row names. Only used if `interactivePlot` is `FALSE`.

Value

If `interactivePlot` is `TRUE`, a `plotly` object. If `interactivePlot` is `FALSE`, a `ggplot2` object.

Author(s)

Charlotte Soneson

Examples

```
library(SummarizedExperiment)
se <- readRDS(system.file("extdata", "GSE102901_cis_se.rds",
                          package = "mutscan"))[1:200, ]
design <- model.matrix(~ Replicate + Condition,
                       data = colData(se))
res <- calculateRelativeFC(se, design, coef = "Conditioncis_output")
plotVolcano(res, pointSize = "large", nTopToLabel = 3)
```

`relabelMutPositions` *Relabel the positions of mutations in the designated ID*

Description

Relabel the positions of mutations in the designated ID

Usage

```
relabelMutPositions(se, conversionTable, mutNameDelimiter = ".")
```

Arguments

`se`	SummarizedExperiment object, with row names of the form XX{.}AA{.}NNN, where XX is the name of the reference sequence, AA is the position of the mutated codon, and NNN is the mutated codon or amino acid. {.} is the delimiter, to be specified in the `mutNameDelimiter` argument. For rows corresponding to sequences with multiple mutated codons, the row names contain multiple names of the form above in a single string, separated by "_".
`conversionTable`	`data.frame` with at least three columns: • seqname The reference sequence name (should match XX in the mutation name) • position The codon position (should match AA in the mutation name) • name The new name for the codon (will replace AA in the mutation name, if the reference sequence matches seqname)
`mutNameDelimiter`	The delimiter used in the mutation name ({.} above).

Value

A SummarizedExperiment object with modified row names.

Author(s)

Charlotte Soneson

Examples

```
x <- readRDS(system.file("extdata", "GSE102901_cis_se.rds",
                         package = "mutscan"))
conversionTable <- data.frame(seqname = "f", position = 0:32)
conversionTable$name = paste0((conversionTable$position - 1) %/% 7 + 1,
                              c("", rep(letters[1:7], 6))[1:33])
out <- relabelMutPositions(x, conversionTable)
```

`summarizeExperiment` *Summarize and collapse multiple mutational scanning experiments*

Description

Combine multiple sequence lists (as returned by `digestFastqs` into a `SummarizedExperiment`, with observed variable sequences (sequence pairs) in rows and samples in columns.

Usage

```
summarizeExperiment(x, coldata, countType = "umis")
```

Arguments

`x`	A named list of objects returned by `digestFastqs`. Names are used to link the objects to the metadata provided in `coldata`.
`coldata`	A `data.frame` with at least one column "Name", which will be used to link to objects in `x`. A potentially subset and reordered version of `coldata` is stored in the `colData` of the returned `SummarizedExperiment`.
`countType`	Either "reads" or "umis". If "reads", the "count" assay of the returned object will contain the observed number of reads for each sequence (pair). If "umis", the "count" assay will contain the number of unique UMIs observed for each sequence (pair).

Value

A `SummarizedExperiment` `x` with

assays(x)$counts containing the observed number of sequences or sequence pairs (if `countType` = "reads"), or the observed number of unique UMIs for each sequence or sequence pair (if `countType` = "umis").

rowData(x) containing the unique sequences or sequence pairs.

colData(x) containing the metadata provided by `coldata`.

Author(s)

Michael Stadler, Charlotte Soneson

Examples

```
## Input sample
inp <- digestFastqs(
    fastqForward = system.file("extdata", "cisInput_1.fastq.gz",
                               package = "mutscan"),
    elementsForward = "SUCV", elementLengthsForward = c(1, 10, 18, 96),
    constantForward = "AACCGGAGGAGGGAGCTG",
    wildTypeForward = c(FOS = paste0(
        "ACTGATACACTCCAAGCGGAGACAGACCAACTAGAAGATGAGAAGTC",
        "TGCTTTGCAGACCGAGATTGCCAACCTGCTGAAGGAGAAGGAAAAACTA")),
    nbrMutatedCodonsMaxForward = 1
)
## Output sample
outp <- digestFastqs(
    fastqForward = system.file("extdata", "cisOutput_1.fastq.gz",
                               package = "mutscan"),
    elementsForward = "SUCV", elementLengthsForward = c(1, 10, 18, 96),
    constantForward = "AACCGGAGGAGGGAGCTG",
    wildTypeForward = c(FOS = paste0(
        "ACTGATACACTCCAAGCGGAGACAGACCAACTAGAAGATGAGAAGTC",
        "TGCTTTGCAGACCGAGATTGCCAACCTGCTGAAGGAGAAGGAAAAACTA")),
    nbrMutatedCodonsMaxForward = 1
)
## Combine
se <- summarizeExperiment(
    x = list(r1inp = inp, r1outp = outp),
    coldata = data.frame(Name = c("r1inp", "r1outp"),
                         Condition = c("input", "output"),
                         Replicate = c("rep1", "rep1")),
    countType = "umis"
)
se
```

Index

calcNearestStringDist, 2
calculateFitnessScore, 3
calculateRelativeFC, 4
collapseMutants
    *(*collapseMutantsBySimilarity*)*,
    6
collapseMutantsByAA
    *(*collapseMutantsBySimilarity*)*,
    6
collapseMutantsBySimilarity, 6

digestFastqs, 8, *28*

generateQCReport, 15
groupSimilarSequences, 17

IUPAC_CODE_MAP, *11*

linkMultipleVariants, 18

plotDistributions, 20
plotFiltering, 21
plotMeanDiff, 22
plotPairs, 23
plotTotals, 25
plotVolcano, 26

relabelMutPositions, 27
render, *16*

SummarizedExperiment, *6, 7*, *28*
summarizeExperiment, *3*, *6*, 28