Package 'MotifPeeker'

February 15, 2025

Type Package

Title Benchmarking Epigenomic Profiling Methods Using Motif Enrichment

Version 0.99.13

Description MotifPeeker is used to compare and analyse datasets from epigenomic profiling methods with motif enrichment as the key benchmark. The package outputs an HTML report consisting of three sections: (1. General Metrics) Overview of peaks-related general metrics for the datasets (FRiP scores, peak widths and motif-summit distances). (2. Known Motif Enrichment Analysis) Statistics for the frequency of user-provided motifs enriched in the datasets. (3. De-Novo Motif Enrichment Analysis) Statistics for the frequency of de-novo discovered motifs enriched in the datasets and compared with known motifs.

License GPL (>= 3)

URL https://github.com/neurogenomics/MotifPeeker

BugReports https://github.com/neurogenomics/MotifPeeker/issues

Depends R (>= 4.4.0)

Imports BiocFileCache, BiocParallel, DT, ggplot2, plotly, universalmotif, GenomicRanges, IRanges, rtracklayer, tools, htmltools, rmarkdown, viridis, SummarizedExperiment, htmlwidgets, Rsamtools, GenomicAlignments, GenomeInfoDb, Biostrings, BSgenome, memes, S4Vectors, dplyr, purrr, tidyr, heatmaply, stats, utils

Suggests BSgenome.Hsapiens.UCSC.hg19, BSgenome.Hsapiens.UCSC.hg38, downloadthis, knitr, markdown, methods, remotes, rworkflows, testthat (>= 3.0.0), withr, emoji, curl, jsonlite

VignetteBuilder knitr

biocViews Epigenetics, Genetics, QualityControl, ChIPSeq, MultipleComparison, FunctionalGenomics, MotifDiscovery, SequenceMatching, Software, Alignment

Config/testthat/edition 3

Encoding UTF-8

LazyData FALSE

RoxygenNote 7.3.2

SystemRequirements MEME Suite (v5.3.3 or above)
<http://meme-suite.org/doc/download.html>

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

git_branch devel

git_last_commit 665ef16

git_last_commit_date 2025-01-06

Repository Bioconductor 3.21

Date/Publication 2025-02-14

Author Hiranyamaya Dash [cre, aut] (ORCID:
<https://orcid.org/0009-0005-5514-505X>),
Thomas Roberts [aut] (ORCID: <https://orcid.org/0009-0006-6244-8670>),
Maria Weinert [aut] (ORCID: <https://orcid.org/0000-0001-6187-1000>),
Nathan Skene [aut] (ORCID: <https://orcid.org/0000-0002-6807-3180>)

Maintainer Hiranyamaya Dash <hdash.work@gmail.com>

Contents

bpapply . . . 3
calc_frip . . . 4
check_dep . . . 5
check_duplicates . . . 6
check_ENCODE . . . 6
check_genome_build . . . 7
check_input . . . 8
check_JASPAR . . . 8
confirm_meme_install . . . 9
CTCF_ChIP_peaks . . . 9
CTCF_TIP_peaks . . . 10
denovo_motifs . . . 10
download_button . . . 12
dt_enrichment_individual . . . 13
filter_repeats . . . 14
find_motifs . . . 15
format_exptype . . . 16
get_df_distances . . . 17
get_df_enrichment . . . 19
get_download_buttons . . . 21
get_JASPARCORE . . . 22
link_JASPAR . . . 23
markov_background_model . . . 23
messager . . . 24
MotifPeeker . . . 24
motif_enrichment . . . 28

motif_MA1102.3 . . . 30
motif_MA1930.2 . . . 30
motif_similarity . . . 31
normalise_paths . . . 34
plot_enrichment_individual . . . 34
plot_enrichment_overall . . . 36
plot_motif_comparison . . . 37
pretty_number . . . 38
print_denovo_sections . . . 38
print_DT . . . 39
print_labels . . . 42
random_string . . . 42
read_motif_file . . . 43
read_peak_file . . . 44
read_peak_file_macs . . . 45
read_peak_file_seacr . . . 46
report_command . . . 46
report_header . . . 47
save_peak_file . . . 47
segregate_seqs . . . 48
summit_to_motif . . . 49
to_plotly . . . 51
trim_seqs . . . 52
use_cache . . . 53
%>% . . . 54

Index **55**

`bpapply` *Use BiocParallel functions with appropriate parameters*

Description

Light wrapper around `BiocParallel` functions that automatically applies appropriate parallel function.

Usage

```
bpapply(
  X,
  FUN,
  apply_fun = BiocParallel::bplapply,
  BPPARAM = BiocParallel::bpparam(),
  progressbar = FALSE,
  force_snowparam = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

X	Any object for which methods length, [, and [[are implemented.
FUN	The function to be applied to each element of X.
apply_fun	A BiocParallel function to use for parallel processing. (default = BiocParallel::bplapply)
BPPARAM	A BiocParallelParam-class object specifying run parameters. (default = bpparam())
...	Arguments passed on to BiocParallel::bplapply, BiocParallel::bpmapply BPREDO A list of output from bplapply with one or more failed elements. When a list is given in BPREDO, bpok is used to identify errors, tasks are rerun and inserted into the original results. BPOPTIONS Additional options to control the behavior of the parallel evaluation, see bpoptions. MoreArgs List of additional arguments to FUN. SIMPLIFY If TRUE the result will be simplified using simplify2array. USE.NAMES If TRUE the result will be named.

Value

Output relevant to the apply_fun specified.

Examples

```
half_it <- function(arg1) return(arg1 / 2)
x <- seq_len(10)

res <- MotifPeeker:::bpapply(x, half_it)
print(res)
```

---

calc_frip *Calculate FRiP score*

Description

Calculate the Fraction of Reads in Peak score from the read and peak file of an experiment.

Usage

```
calc_frip(read_file, peak_file, single_end = TRUE, total_reads = NULL)
```

Arguments

read_file	A BamFile object.
peak_file	A GRanges object.
single_end	A logical value. If TRUE, the reads classified as single-ended. (default = TRUE)
total_reads	(optional) The total number of reads in the experiment. Skips counting the total number of reads if provided, saving computation.

Details

The FRiP score is calculated as follows:

$$\text{FRiP} = \frac{(\text{number of reads in peaks})}{(\text{total number of reads})}$$

Value

A numeric value indicating the FRiP score.

Examples

```
read_file <- system.file("extdata", "CTCF_ChIP_alignment.bam",
                         package = "MotifPeeker")
read_file <- Rsamtools::BamFile(read_file)
data("CTCF_ChIP_peaks", package = "MotifPeeker")

calc_frip(read_file, CTCF_ChIP_peaks)
```

---

check_dep *Check attached dependency*

Description

Stop execution if a package is not attached.

Usage

```
check_dep(pkg, fatal = TRUE, custom_msg = NULL)
```

Arguments

pkg	a character string of the package name
fatal	a logical value indicating whether to stop execution if the package is not attached.
custom_msg	a custom message to display if the package is not attached.

Value

TRUE if the package is available or else FALSE if `fatal = FALSE`.

check_duplicates *Check for duplicates*

Description

Checks for duplicated items in a vector or list and throw an error if found.

Usage

```
check_duplicates(x)
```

Arguments

x	A vector or list.

Value

Null

check_ENCODE *Check for ENCODE input*

Description

Check and get files from ENCODE project. Requires the input to be in ENCODE ID format. Uses BiocFileCache to cache downloads. Only works for files.

Usage

```
check_ENCODE(encode_id, expect_format, verbose = FALSE)
```

Arguments

encode_id	A character string specifying the ENCODE ID.
expect_format	A character string (or a vector) specifying the expected format(s) of the file. If the file is not in the expected format, an error is thrown.
verbose	A logical indicating whether to print verbose messages while running the function. (default = FALSE)

Value

A character string specifying the path to the downloaded file.

Examples

```
if (requireNamespace("curl", quietly = TRUE) &&
    requireNamespace("jsonlite", quietly = TRUE)) {
    check_ENCODE("ENCFF920TXI", expect_format = c("bed", "gz"))
}
```

`check_genome_build` *Check genome build*

Description

Check if the genome build is valid and return the appropriate BSGenome object.

Usage

```
check_genome_build(genome_build)
```

Arguments

`genome_build`	A character string with the abbreviated genome build name, or a BSGenome object. At the moment, only `hg38` and `hg19` are supported as abbreviated input.

Value

A BSGenome object.

See Also

BSgenome-class for more information on BSGenome objects.

Examples

```
if (requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly = TRUE)) {
    check_genome_build("hg38")
}
```

check_input *Check for input validity and pass to appropriate function*

Description

Check for input validity and pass to appropriate function

Usage

```
check_input(x, type, FUN, inverse = FALSE, ...)
```

Arguments

x	The input to check.
type	The type of input to check for. Supported types are: • jaspar_id: JASPAR identifier. • motif: 'universalmotif' motif object. • encode_id: ENCODE identifier.
FUN	The function to pass the input to.
inverse	Logical indicating whether to return the input if it is invalid for the specified 'type'.
...	Additional arguments to pass to the 'FUN' function.

Value

'x' if the input is invalid for the specified 'type', or else the output of the 'FUN' function. If 'inverse = TRUE', the function returns the output of the 'FUN' function if the input is valid, or else 'x'.

check_JASPAR *Check for JASPAR input*

Description

Check and get files from JASPAR. Requires the input to be in JASPAR ID format. Uses BiocFileCache to cache downloads.

Usage

```
check_JASPAR(motif_id, verbose = FALSE)
```

Arguments

motif_id	A character string specifying the JASPAR motif ID.
verbose	A logical indicating whether to print verbose messages while running the function. (default = FALSE)

Value

A character string specifying the path to the downloaded file.

Examples

```
check_JASPAR("MA1930.2")
```

`confirm_meme_install` *Stop if MEME suite is not installed*

Description

Stop if MEME suite is not installed

Usage

```
confirm_meme_install(meme_path = NULL, continue = FALSE)
```

Arguments

`meme_path`	path to `meme/bin/` (optional). Defaut: `NULL`, searches "MEME_PATH" environment variable or "meme_path" option for path to "meme/bin/".
`continue`	Continue code execution if MEME suite is not installed.

Value

Null

See Also

`check_meme_install`

`CTCF_ChIP_peaks` *Example ChIP-seq peak file*

Description

Human CTCF peak file generated with ChIP-seq using HCT116 cell-line. No control files were used to generate the peak file.

Usage

```
data("CTCF_ChIP_peaks")
```

Format

An object of class GRanges of length 209.

Note

To reduce the size of the package, the included peak file focuses on specific genomic regions. The subset region included is chr10:65,654,529-74,841,155.

Source

ENCODE Accession: ENCFF091ODJ

CTCF_TIP_peaks *Example TIP-seq peak file*

Description

Human CTCF peak file generated with TIP-seq using HCT116 cell-line. The peak file was generated using the nf-core/cutandrun pipeline. Raw read files were sourced from *NIH Sequence Read Archives* (ID: SRR16963166).

Usage

```
data("CTCF_TIP_peaks")
```

Format

An object of class GRanges of length 182.

Note

To reduce the size of the package, the included peak file focuses on specific genomic regions. The subset region included is chr10:65,654,529-74,841,155.

---

denovo_motifs *Discover motifs in sequences*

Description

Use STREME from MEME suite to find motifs in the provided sequences. To speed up the process, the sequences can be optionally trimmed to reduce the search space. The result is then optionally filtered to remove motifs with a high number of nucleotide repeats

Usage

```
denovo_motifs(
  seqs,
  trim_seq_width,
  genome_build,
  discover_motifs_count = 3,
  minw = 8,
  maxw = 25,
  filter_n = 6,
  out_dir = tempdir(),
  meme_path = NULL,
  BPPARAM = BiocParallel::SerialParam(),
  verbose = FALSE,
  debug = FALSE,
  ...
)
```

Arguments

seqs	A list of GRanges objects containing sequences to search for motifs.
trim_seq_width	An integer specifying the width of the sequence to extract around the summit (default = NULL). This sequence is used to search for discovered motifs. If not provided, the entire peak region will be used. This parameter is intended to reduce the search space and speed up motif discovery; therefore, a value less than the average peak width is recommended. Peaks are trimmed symmetrically around the summit while respecting the peak bounds.
genome_build	The genome build that the peak sequences should be derived from.
discover_motifs_count	An integer specifying the number of motifs to discover. (default = 3) Note that higher values take longer to compute.
minw	An integer specifying the minimum width of the motif. (default = 8)
maxw	An integer specifying the maximum width of the motif. (default = 25)
filter_n	An integer specifying the number of consecutive nucleotide repeats a discovered motif must contain to be filtered out. (default = 6)
out_dir	A character vector of output directory to save STREME results to. (default = tempdir())
meme_path	path to "meme/bin/" (default: NULL). Will use default search behavior as described in check_meme_install() if unset.
BPPARAM	A BiocParallelParam-class object specifying run parameters. (default = SerialParam(), single core run)
verbose	A logical indicating whether to print verbose messages while running the function. (default = FALSE)
debug	A logical indicating whether to print debug messages while running the function. (default = FALSE)
...	Additional arguments to pass to STREME. For more information, refer to the official MEME Suite documentation on STREME.

Value

A list of universalmotif objects and associated metadata.

Examples

```
if (memes::meme_is_installed()) {
data("CTCF_TIP_peaks", package = "MotifPeeker")
if (requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly = TRUE)) {
    genome_build <- BSgenome.Hsapiens.UCSC.hg38::BSgenome.Hsapiens.UCSC.hg38

    res <- denovo_motifs(list(CTCF_TIP_peaks),
                         trim_seq_width = 50,
                         genome_build = genome_build,
                         discover_motifs_count = 1,
                         filter_n = 6,
                         minw = 8,
                         maxw = 8,
                         out_dir = tempdir())
    print(res[[1]]$consensus)
}
}
```

---

download_button *Create a download button*

Description

Creates a download button for a file or directory, suitable to embed into an HTML report.

Usage

```
download_button(
  path,
  type,
  button_label,
  output_name = NULL,
  button_type = "success",
  has_icon = TRUE,
  icon = "fa fa-save",
  add_button = TRUE,
  ...
)
```

Arguments

path	A character string specifying the path to the file or directory.
type	A character string specifying the type of download. Either "file" or "dir".
button_label	Character (HTML), button label

output_name	Name of of the output file. If not specified, it will take the source file's name if one file is specified. In case of multiple files, the output_name must be specified.
button_type	Character, one of the standard Bootstrap types
has_icon	Specify whether to include fontawesome icons in the button label
icon	Fontawesome tag e.g.: "fa fa-save"
add_button	A logical indicating whether to add the download button to the HTML report. (default = TRUE)
...	Arguments passed on to downloadthis::download_file self_contained A boolean to specify whether your HTML output is self-contained. Default to FALSE.

Value

htmltools::tag, <button>

See Also

download_file

---

dt_enrichment_individual *Get* datatable *for motif-enrichment of individual experiments.*

Description

Get datatable for motif-enrichment of individual experiments.

Usage

```
dt_enrichment_individual(
  result,
  enrichment_df,
  comparison_i,
  motif_i,
  reference_index = 1
)
```

Arguments

result	A list with the following elements: **peaks** A list of peak files generated using read_peak_file. **alignments** A list of alignment files. **exp_type** A character vector of experiment types. **exp_labels** A character vector of experiment labels.

read_count A numeric vector of read counts.

peak_count A numeric vector of peak counts.

enrichment_df	A data frame containing the motif enrichment results, produced using get_df_enrichment.
comparison_i	The index of the comparison dataset to plot.
motif_i	The index of the motif to plot.
reference_index	An integer specifying the index of the peak file to use as the reference dataset for comparison. Indexing starts from 1. (default = 1)

Value

A DT::datatable object with the peak motif enrichment data for the specified comparison_i and motif_i.

See Also

Other datatable functions: print_denovo_sections()

---

filter_repeats *Filter motifs with nucleotide repeats*

Description

Filter out motifs which contain filter_n or more consecutive nucleotide repeats. This includes unambiguous bases such as 'Y', 'N, 'R', etc.

Usage

```
filter_repeats(motifs, filter_n = 6)
```

Arguments

motifs	Output from runStreme.
filter_n	Minimum number of consecutive nucleotide repeats to filter.

Value

A list object with same structure as motifs but with motifs containing filter_n or more consecutive nucleotide repeats removed.

See Also

runStreme

find_motifs *Find similar motifs*

Description

Search through provided motif database to find similar motifs to the input. Light wrapper around TOMTOM from MEME Suite.

Usage

```
find_motifs(
  streme_out,
  motif_db,
  out_dir = tempdir(),
  meme_path = NULL,
  BPPARAM = BiocParallel::bpparam(),
  verbose = FALSE,
  debug = FALSE,
  ...
)
```

Arguments

streme_out	Output from denovo_motifs.
motif_db	Path to .meme format file to use as reference database, or a list of universalmotif-class objects. (optional) Results from de-novo motif discovery are searched against this database to find similar motifs. If not provided, JASPAR CORE database will be used. **NOTE**: p-value estimates are inaccurate when the database has fewer than 50 entries.
out_dir	A character vector of output directory to save STREME results to. (default = tempdir())
meme_path	path to "meme/bin/" (default: NULL). Will use default search behavior as described in check_meme_install() if unset.
BPPARAM	A BiocParallelParam-class object specifying run parameters. (default = bpparam())
verbose	A logical indicating whether to print verbose messages while running the function. (default = FALSE)
debug	A logical indicating whether to print debug messages while running the function. (default = FALSE)
...	Additional arguments to pass to TOMTOM. For more information, refer to the official MEME Suite documentation on TOMTOM.

Value

data.frame of match results. Contains `best_match_motif` column of `universalmotif` objects with the matched PWM from the database, a series of `best_match_*` columns describing the TomTom results of the match, and a `tomtom` list column storing the ranked list of possible matches to each motif. If a universalmotif data.frame is used as input, these columns are appended to the data.frame. If no matches are returned, `tomtom` and `best_match_motif` columns will be set to `NA` and a message indicating this will print.

Examples

```
if (memes::meme_is_installed()) {
    data("CTCF_TIP_peaks", package = "MotifPeeker")

    if (requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly = TRUE)) {
        genome_build <-
            BSgenome.Hsapiens.UCSC.hg38::BSgenome.Hsapiens.UCSC.hg38

        res <- denovo_motifs(list(CTCF_TIP_peaks),
                        trim_seq_width = 50,
                        genome_build = genome_build,
                        discover_motifs_count = 1,
                        filter_n = 10,
                        out_dir = tempdir())
        res2 <- find_motifs(res, motif_db = get_JASPARCORE(),
                            out_dir = tempdir())
        print(res2)
    }
}
```

---

format_exptype *Format exp_type*

Description

Format input exp_type to look pretty.

Usage

```
format_exptype(exp_type)
```

Arguments

`exp_type`	A character depicting the type of experiment. Supported experimental types are: • `chipseq`: ChIP-seq data • `tipseq`: TIP-seq data • `cuttag`: CUT&Tag data • `cutrun`: CUT&Run data

- other: Other experiment type data
- unknown: Unknown experiment type data

Any item not mentioned above will be returned as-is.

Value

A character vector of formatted exp_type.

Examples

```
MotifPeeker:::format_exptype("chipseq")
```

`get_df_distances` *Get dataframe with motif-summit distances*

Description

Wrapper for 'MotifPeeker::summit_to_motif' to get motif-summit distances for all peaks and motifs, generating a `data.frame` suitable for plots.

Usage

```
get_df_distances(
  result,
  user_motifs,
  genome_build,
  out_dir = tempdir(),
  BPPARAM = BiocParallel::bpparam(),
  meme_path = NULL,
  verbose = FALSE
)
```

Arguments

`result`	A `list` with the following elements: **peaks** A `list` of peak files generated using `read_peak_file`. **alignments** A `list` of alignment files. **exp_type** A `character` vector of experiment types. **exp_labels** A `character` vector of experiment labels. **read_count** A `numeric` vector of read counts. **peak_count** A `numeric` vector of peak counts.
`user_motifs`	A `list` with the following elements: **motifs** A `list` of motif files. **motif_labels** A `character` vector of motif labels.

genome_build
: A character string with the abbreviated genome build name, or a BSGenome object. At the moment, only `hg38` and `hg19` are supported as abbreviated input.

out_dir
: A `character` vector of output directory.

BPPARAM
: A `BiocParallelParam-class` object enabling parallel execution. (default = SerialParam(), single-CPU run)

  Following are two examples of how to set up parallel processing:

  - `BPPARAM = BiocParallel::MulticoreParam(4)`: Uses 4 CPU cores for parallel processing.
  - `library("BiocParallel")` followed by `register(MulticoreParam(4))` sets all subsequent BiocParallel functions to use 4 CPU cores. `Motifpeeker()` must be run with `BPPARAM = BiocParallel::MulticoreParam()`.

  IMPORTANT: For each worker, please ensure a minimum of 8GB of memory (RAM) is available as `motif_discovery` is memory-intensive.

meme_path
: path to `meme/bin/` (optional). Defaut: `NULL`, searches "MEME_PATH" environment variable or "meme_path" option for path to "meme/bin/".

verbose
: A logical indicating whether to print verbose messages while running the function. (default = FALSE)

Value

A `data.frame` with the following columns:

exp_label Experiment labels.

exp_type Experiment types.

motif_indice Motif indices.

distance Distances between peak summit and motif.

See Also

Other generate data.frames: `get_df_enrichment()`

Examples

```
if (memes::meme_is_installed()) {
data("CTCF_ChIP_peaks", package = "MotifPeeker")
data("motif_MA1102.3", package = "MotifPeeker")
data("motif_MA1930.2", package = "MotifPeeker")
input <- list(
    peaks = CTCF_ChIP_peaks,
    exp_type = "ChIP",
    exp_labels = "CTCF",
    read_count = 150,
    peak_count = 100
)
motifs <- list(
    motifs = list(motif_MA1930.2, motif_MA1102.3),
    motif_labels = list("MA1930.2", "MA1102.3")
```

```
)

if (requireNamespace("BSgenome.Hsapiens.UCSC.hg38")) {
    genome_build <- BSgenome.Hsapiens.UCSC.hg38::BSgenome.Hsapiens.UCSC.hg38
    distances_df <- get_df_distances(input, motifs, genome_build)
    print(distances_df)
}
}
```

`get_df_enrichment` *Get dataframe with motif enrichment values*

Description

Wrapper for 'MotifPeeker::motif_enrichment' to get motif enrichment counts and percentages for all peaks and motifs, generating a `data.frame` suitable for plots. The `data.frame` contains values for all and segregated peaks.

Usage

```
get_df_enrichment(
  result,
  segregated_peaks,
  user_motifs,
  genome_build,
  reference_index = 1,
  out_dir = tempdir(),
  BPPARAM = BiocParallel::bpparam(),
  meme_path = NULL,
  verbose = FALSE
)
```

Arguments

`result`	A `list` with the following elements: **peaks** A `list` of peak files generated using `read_peak_file`. **alignments** A `list` of alignment files. **exp_type** A `character` vector of experiment types. **exp_labels** A `character` vector of experiment labels. **read_count** A `numeric` vector of read counts. **peak_count** A `numeric` vector of peak counts.
`segregated_peaks`	A `list` object generated using `segregate_seqs`.
`user_motifs`	A `list` with the following elements: **motifs** A `list` of motif files.

motif_labels A `character` vector of motif labels.

`genome_build` A character string with the abbreviated genome build name, or a BSGenome object. At the moment, only `hg38` and `hg19` are supported as abbreviated input.

`reference_index`
An integer specifying the index of the peak file to use as the reference dataset for comparison. Indexing starts from 1. (default = 1)

`out_dir` A `character` vector of output directory.

`BPPARAM` A `BiocParallelParam-class` object enabling parallel execution. (default = SerialParam(), single-CPU run)

Following are two examples of how to set up parallel processing:

- `BPPARAM = BiocParallel::MulticoreParam(4)`: Uses 4 CPU cores for parallel processing.
- `library("BiocParallel")` followed by `register(MulticoreParam(4))` sets all subsequent BiocParallel functions to use 4 CPU cores. `Motifpeeker()` must be run with `BPPARAM = BiocParallel::MulticoreParam()`.

IMPORTANT: For each worker, please ensure a minimum of 8GB of memory (RAM) is available as `motif_discovery` is memory-intensive.

`meme_path` path to `meme/bin/` (optional). Defaut: `NULL`, searches "MEME_PATH" environment variable or "meme_path" option for path to "meme/bin/".

`verbose` A logical indicating whether to print verbose messages while running the function. (default = FALSE)

Value

A `data.frame` with the following columns:

exp_label Experiment labels.

exp_type Experiment types.

motif_indice Motif indices.

group1 Segregated group- "all", "Common" or "Unique".

group2 "reference" or "comparison" group.

count_enriched Number of peaks with motif.

count_nonenriched Number of peaks without motif.

perc_enriched Percentage of peaks with motif.

perc_nonenriched Percentage of peaks without motif.

See Also

Other generate data.frames: `get_df_distances()`

Examples

```
if (memes::meme_is_installed()) {
    data("CTCF_ChIP_peaks", package = "MotifPeeker")
    data("CTCF_TIP_peaks", package = "MotifPeeker")
    data("motif_MA1102.3", package = "MotifPeeker")
    data("motif_MA1930.2", package = "MotifPeeker")
    input <- list(
        peaks = list(CTCF_ChIP_peaks, CTCF_TIP_peaks),
        exp_type = c("ChIP", "TIP"),
        exp_labels = c("CTCF_ChIP", "CTCF_TIP"),
        read_count = c(150, 200),
        peak_count = c(100, 120)
    )
    segregated_input <- segregate_seqs(input$peaks[[1]], input$peaks[[2]])
    motifs <- list(
        motifs = list(motif_MA1930.2, motif_MA1102.3),
        motif_labels = list("MA1930.2", "MA1102.3")
    )
    reference_index <- 1

    if (requireNamespace("BSgenome.Hsapiens.UCSC.hg38")) {
        genome_build <-
            BSgenome.Hsapiens.UCSC.hg38::BSgenome.Hsapiens.UCSC.hg38

        enrichment_df <- get_df_enrichment(
            input, segregated_input, motifs, genome_build,
            reference_index = 1
        )
    }
}
```

---

get_download_buttons *Get download buttons for peak file, STREME and TOMOTM output*

Description

Get download buttons for peak file, STREME and TOMOTM output

Usage

```
get_download_buttons(
  comparison_i,
  start_i,
  segregated_peaks,
  out_dir,
  add_buttons = TRUE,
  verbose = FALSE
)
```

Arguments

comparison_i	Index of the comparison pair group.
start_i	Index of the first comparison pair.
segregated_peaks	A list of peak files generated from segregate_seqs.
out_dir	A character vector of the directory with STREME and TOMTOM output.
add_buttons	A logical indicating whether to prepare download buttons.
verbose	A logical indicating whether to print messages.

Value

A list of download buttons for peak file, STREME and TOMTOM output.

---

get_JASPARCORE *Download JASPAR CORE database*

Description

Downloads JASPAR CORE database in meme format for all available taxonomic groups. Uses BiocFileCache to cache downloads.

Usage

```
get_JASPARCORE(verbose = FALSE)
```

Arguments

verbose	A logical indicating whether to print verbose messages while running the function. (default = FALSE)

Value

A character string specifying the path to the downloaded file (meme format).

Examples

```
get_JASPARCORE()
```

link_JASPAR *Get JASPAR link for motifs*

Description

Get JASPAR link for motifs

Usage

```
link_JASPAR(motif_id, download = FALSE)
```

Arguments

motif_id	A character string specifying the JASPAR motif ID.
download	A logical specifying whether to return a download link or an HTML embeddable matrix profile link. (default = FALSE)

Value

A character string containing the JASPAR motif link.

markov_background_model *Generate a 0-order Markov background model*

Description

`markov_background_model()` generates a 0-order background model for use with FIMO or AME. The function uses the letter frequencies in the input sequences to generate the background model.

Usage

```
markov_background_model(sequences, out_dir, verbose = FALSE)
```

Arguments

sequences	A DNAStringSet object.
out_dir	Location to save the 0-order background file.
verbose	A logical indicating whether to print verbose messages while running the function. (default = FALSE)

Value

The path to the 0-order background file.

messager *Print messages*

Description

Conditionally print messages. Allows developers to easily control verbosity of functions, and meet Bioconductor requirements that dictate the message must first be stored to a variable before passing to message.

Usage

```
messager(..., v = Sys.getenv("VERBOSE") != "FALSE")
```

Arguments

v	Whether to print messages or not.

Value

Null

MotifPeeker *Benchmark epigenomic profiling methods using motif enrichment*

Description

This function compares different epigenomic datasets using motif enrichment as the key metric. The output is an easy-to-interpret HTML document with the results. The report contains three main sections: (1) General Metrics on peak and alignment files (if provided), (2) Known Motif Enrichment Analysis and (3) Discovered Motif Enrichment Analysis.

Usage

```
MotifPeeker(
  peak_files,
  reference_index = 1,
  alignment_files = NULL,
  exp_labels = NULL,
  exp_type = NULL,
  genome_build,
  motif_files = NULL,
  motif_labels = NULL,
  cell_counts = NULL,
  motif_discovery = TRUE,
  motif_discovery_count = 3,
  filter_n = 6,
```

```
  trim_seq_width = NULL,
  motif_db = NULL,
  download_buttons = TRUE,
  meme_path = NULL,
  out_dir = tempdir(),
  save_runfiles = FALSE,
  display = if (interactive()) "browser",
  BPPARAM = BiocParallel::SerialParam(),
  quiet = TRUE,
  debug = FALSE,
  verbose = FALSE
)
```

Arguments

`peak_files` A character vector of path to peak files, or a vector of GRanges objects generated using `read_peak_file`. Currently, peak files from the following peak-calling tools are supported:

- MACS2: `.narrowPeak` files
- SEACR: `.bed` files

ENCODE file IDs can also be provided to automatically fetch peak file(s) from the ENCODE database.

`reference_index` An integer specifying the index of the peak file to use as the reference dataset for comparison. Indexing starts from 1. (default = 1)

`alignment_files` A character vector of path to alignment files, or a vector of `BamFile` objects. (optional) Alignment files are used to calculate read-related metrics like FRiP score. ENCODE file IDs can also be provided to automatically fetch alignment file(s) from the ENCODE database.

`exp_labels` A character vector of labels for each peak file. (optional) If not provided, capital letters will be used as labels in the report.

`exp_type` A character vector of experimental types for each peak file. (optional) Useful for comparison of different methods. If not provided, all datasets will be classified as "unknown" experiment types in the report. Supported experimental types are:

- `chipseq`: ChIP-seq data
- `tipseq`: TIP-seq data
- `cuttag`: CUT&Tag data
- `cutrun`: CUT&Run data

`exp_type` is used only for labelling. It does not affect the analysis. You can also input custom strings. Datasets will be grouped as long as they match their respective `exp_type`.

`genome_build` A character string with the abbreviated genome build name, or a BSGenome object. At the moment, only `hg38` and `hg19` are supported as abbreviated input.

`motif_files` A character vector of path to motif files, or a vector of `universalmotif-class` objects. (optional) Required to run *Known Motif Enrichment Analysis*. JASPAR matrix IDs can also be provided to automatically fetch motifs from the JASPAR.

`motif_labels` A character vector of labels for each motif file. (optional) Only used if path to file names are passed in `motif_files`. If not provided, the motif file names will be used as labels.

`cell_counts` An integer vector of experiment cell counts for each peak file. (optional) Creates additional comparisons based on cell counts.

`motif_discovery`
A logical indicating whether to perform motif discovery for the third section of the report. (default = TRUE)

`motif_discovery_count`
An integer specifying the number of motifs to discover. (default = 3) Note that higher values take longer to compute.

`filter_n` An integer specifying the number of consecutive nucleotide repeats a discovered motif must contain to be filtered out. (default = 6)

`trim_seq_width` An integer specifying the width of the sequence to extract around the summit (default = NULL). This sequence is used to search for discovered motifs. If not provided, the entire peak region will be used. This parameter is intended to reduce the search space and speed up motif discovery; therefore, a value less than the average peak width is recommended. Peaks are trimmed symmetrically around the summit while respecting the peak bounds.

`motif_db` Path to `.meme` format file to use as reference database, or a list of `universalmotif-class` objects. (optional) Results from de-novo motif discovery are searched against this database to find similar motifs. If not provided, JASPAR CORE database will be used. **NOTE**: p-value estimates are inaccurate when the database has fewer than 50 entries.

`download_buttons`
A logical indicating whether to include download buttons for various files within the HTML report. (default = TRUE)

`meme_path` path to `meme/bin/` (optional). Defaut: `NULL`, searches "MEME_PATH" environment variable or "meme_path" option for path to "meme/bin/".

`out_dir` A character string specifying the directory to save the output files. (default = `tempdir()`) A sub-directory with the output files will be created in this directory.

`save_runfiles` A logical indicating whether to save intermediate files generated during the run, such as those from FIMO and AME. (default = FALSE)

`display` A character vector specifying the display mode for the HTML report once it is generated. (default = NULL) Options are:

- `"browser"`: Open the report in the default web browser.
- `"rstudio"`: Open the report in the RStudio Viewer.
- `NULL`: Do not open the report.

`BPPARAM` A `BiocParallelParam-class` object enabling parallel execution. (default = SerialParam(), single-CPU run)

Following are two examples of how to set up parallel processing:

- `BPPARAM = BiocParallel::MulticoreParam(4)`: Uses 4 CPU cores for parallel processing.
- `library("BiocParallel")` followed by `register(MulticoreParam(4))` sets all subsequent BiocParallel functions to use 4 CPU cores. `Motifpeeker()` must be run with `BPPARAM = BiocParallel::MulticoreParam()`.

IMPORTANT: For each worker, please ensure a minimum of 8GB of memory (RAM) is available as `motif_discovery` is memory-intensive.

`quiet` A logical indicating whether to print markdown knit messages. (default = FALSE)

`debug` A logical indicating whether to print debug/error messages in the HTML report. (default = FALSE)

`verbose` A logical indicating whether to print verbose messages while running the function. (default = FALSE)

Details

Runtime guidance: For 4 datasets, the runtime is approximately 3 minutes with motif_discovery disabled. However, motif discovery can take hours to complete. To make computation faster, we highly recommend tuning the following arguments:

`BPPARAM=MulticoreParam(x)` Running motif discovery in parallel can significantly reduce runtime, but it is very memory-intensive, consuming 10+GB of RAM per thread. Memory starvation can greatly slow the process, so set the number of cores with caution.

`motif_discovery_count` The number of motifs to discover per sequence group exponentially increases runtime. We recommend no more than 5 motifs to make a meaningful inference.

`trim_seq_width` Trimming sequences before running motif discovery can significantly reduce the search space. Sequence length can exponentially increase runtime. We recommend running the script with `motif_discovery = FALSE` and studying the motif-summit distance distribution under general metrics to find the sequence length that captures most motifs. A good starting point is 150 but it can be reduced further if appropriate.

Value

Path to the output directory.

Note

Running motif discovery is computationally expensive and can require from minutes to hours. `denovo_motifs` can widely affect the runtime (higher values take longer). Setting `trim_seq_width` to a lower value can also reduce the runtime significantly.

Examples

```
peaks <- list(
    system.file("extdata", "CTCF_ChIP_peaks.narrowPeak",
                package = "MotifPeeker"),
    system.file("extdata", "CTCF_TIP_peaks.narrowPeak",
                package = "MotifPeeker")
)
```

```
alignments <- list(
    system.file("extdata", "CTCF_ChIP_alignment.bam",
                package = "MotifPeeker"),
    system.file("extdata", "CTCF_TIP_alignment.bam",
                package = "MotifPeeker")
)

motifs <- list(
    system.file("extdata", "motif_MA1930.2.jaspar",
                package = "MotifPeeker"),
    system.file("extdata", "motif_MA1102.3.jaspar",
                package = "MotifPeeker")
)

if (memes::meme_is_installed()) {
    MotifPeeker(
        peak_files = peaks,
        reference_index = 2,
        alignment_files = alignments,
        exp_labels = c("ChIP", "TIP"),
        exp_type = c("chipseq", "tipseq"),
        genome_build = "hg38",
        motif_files = motifs,
        motif_labels = NULL,
        cell_counts = NULL,
        motif_discovery = TRUE,
        motif_discovery_count = 2,
        motif_db = NULL,
        download_buttons = TRUE,
        out_dir = tempdir(),
        debug = FALSE,
        quiet = TRUE,
        verbose = FALSE
    )
}
```

`motif_enrichment` *Calculate motif enrichment in a set of sequences*

Description

`motif_enrichment()` calculates motif enrichment relative to a set of background sequences using Analysis of Motif Enrichment (AME) from memes.

Usage

```
motif_enrichment(
  peak_input,
```

```
  motif,
  genome_build,
  out_dir = tempdir(),
  verbose = FALSE,
  meme_path = NULL,
  ...
)
```

Arguments

peak_input	Either a path to the narrowPeak file or a GRanges peak object generated by `read_peak_file()`.
motif	An object of class `universalmotif`.
genome_build	The genome build that the peak sequences should be derived from.
out_dir	Location to save the 0-order background file along with the AME output files.
verbose	A logical indicating whether to print verbose messages while running the function. (default = FALSE)
meme_path	path to "meme/bin/" (default: `NULL`). Will use default search behavior as described in `check_meme_install()` if unset.
...	Arguments passed on to `memes::runAme` `method` default: fisher (allowed values: fisher, ranksum, pearson, spearman, 3dmhg, 4dmhg) `sequences` `logical(1)` add results from `sequences.tsv` to `sequences` list column to returned data.frame. Valid only if method = "fisher". See AME outputs webpage for more information (Default: FALSE). `silent` whether to suppress stdout (default: TRUE), useful for debugging.

Value

A list containing a AME results data frame and a numeric referring to the proportion of peaks with a motif.

See Also

`runAme`

Examples

```
if (memes::meme_is_installed()) {
    data("CTCF_TIP_peaks", package = "MotifPeeker")
    data("motif_MA1102.3", package = "MotifPeeker")

    res <- motif_enrichment(
        peak_input = CTCF_TIP_peaks,
        motif = motif_MA1102.3,
        genome_build =
            BSgenome.Hsapiens.UCSC.hg38::BSgenome.Hsapiens.UCSC.hg38,
```

```
    )
    print(res)
}
```

`motif_MA1102.3` *Example CTCFL JASPAR motif file*

Description

The motif file contains the JASPAR motif for CTCFL (MA1102.3) for *Homo Sapiens*. This is one of the two motif files used to demonstrate `MotifPeeker`'s known-motif analysis functionality.

Usage

```
data("motif_MA1102.3")
```

Format

An object of class `universalmotif` of length 1.

Source

JASPAR Matrix ID: MA1102.3

`motif_MA1930.2` *Example CTCF JASPAR motif file*

Description

The motif file contains the JASPAR motif for CTCF (MA1930.2) for *Homo Sapiens*. This is one of the two motif files used to demonstrate `MotifPeeker`'s known-motif analysis functionality.

Usage

```
data("motif_MA1930.2")
```

Format

An object of class `universalmotif` of length 1.

Source

JASPAR Matrix ID: MA1930.2

`motif_similarity` *Compare motifs from segregated sequences*

Description

Compute motif similarity scores between motifs discovered from segregated sequences. Wrapper around `compare_motifs` to compare motifs from different groups of sequences. To see the possible similarity measures available, refer to details.

Usage

```
motif_similarity(
  streme_out,
  method = "PCC",
  normalise.scores = TRUE,
  BPPARAM = BiocParallel::bpparam(),
  ...
)
```

Arguments

`streme_out`	Output from `denovo_motifs`.
`method`	`character(1)` One of PCC, EUCL, SW, KL, ALLR, BHAT, HELL, SEUCL, MAN, ALLR_LL, WEUCL, WPCC. See details.
`normalise.scores`	`logical(1)` Favour alignments which leave fewer unaligned positions, as well as alignments between motifs of similar length. Similarity scores are multiplied by the ratio of aligned positions to the total number of positions in the larger motif, and the inverse for distance scores.
`BPPARAM`	A `BiocParallelParam-class` object specifying run parameters. (default = bpparam())
`...`	Arguments passed on to `universalmotif::compare_motifs` `motifs` See `convert_motifs()` for acceptable motif formats. `compare.to` `numeric` If missing, compares all motifs to all other motifs. Otherwise compares all motifs to the specified motif(s). `db.scores` `data.frame` or `DataFrame`. See `details`. `use.freq` `numeric(1)`. For comparing the `multifreq` slot. `use.type` `character(1)` One of `'PPM'` and `'ICM'`. The latter allows for taking into account the background frequencies if `relative_entropy = TRUE`. Note that `'ICM'` is not allowed when `method = c("ALLR", "ALLR_LL")`. `tryRC` `logical(1)` Try the reverse complement of the motifs as well, report the best score. `min.overlap` `numeric(1)` Minimum overlap required when aligning the motifs. Setting this to a number higher then the width of the motifs will not allow any overhangs. Can also be a number between 0 and 1, representing the minimum fraction that the motifs must overlap.

`min.mean.ic` `numeric(1)` Minimum mean information content between the two motifs for an alignment to be scored. This helps prevent scoring alignments between low information content regions of two motifs. Note that this can result in some comparisons failing if no alignment passes the mean IC threshold. Use `average_ic()` to filter out low IC motifs to get around this if you want to avoid getting `NA`s in your output.

`min.position.ic` `numeric(1)` Minimum information content required between individual alignment positions for it to be counted in the final alignment score. It is recommended to use this together with `normalise.scores = TRUE`, as this will help punish scores resulting from only a fraction of an alignment.

`relative_entropy` `logical(1)` Change the ICM calculation affecting `min.position.ic` and `min.mean.ic`. See `convert_type()`.

`max.p` `numeric(1)` Maximum P-value allowed in reporting matches. Only used if `compare.to` is set.

`max.e` `numeric(1)` Maximum E-value allowed in reporting matches. Only used if `compare.to` is set. The E-value is the P-value multiplied by the number of input motifs times two.

`nthreads` `numeric(1)` Run `compare_motifs()` in parallel with `nthreads` threads. `nthreads = 0` uses all available threads.

`score.strat` `character(1)` How to handle column scores calculated from motif alignments. "sum": add up all scores. "a.mean": take the arithmetic mean. "g.mean": take the geometric mean. "median": take the median. "wa.mean", "wg.mean": weighted arithmetic/geometric mean. "fzt": Fisher Z-transform. Weights are the total information content shared between aligned columns.

`output.report` `character(1)` Provide a filename for `compare_motifs()` to write an html ouput report to. The top matches are shown alongside figures of the match alignments. This requires the `knitr` and `rmarkdown` packages. (Note: still in development.)

`output.report.max.print` `numeric(1)` Maximum number of top matches to print.

Details

Available metrics:

The following metrics are available:

- Euclidean distance (`EUCL`) (Choi et al. 2004)
- Weighted Euclidean distance (`WEUCL`)
- Kullback-Leibler divergence (`KL`) (Kullback and Leibler 1951; Roepcke et al. 2005)
- Hellinger distance (`HELL`) (Hellinger 1909)
- Squared Euclidean distance (`SEUCL`)
- Manhattan distance (`MAN`)
- Pearson correlation coefficient (`PCC`)
- Weighted Pearson correlation coefficient (`WPCC`)

- Sandelin-Wasserman similarity (SW), or sum of squared distances (Sandelin and Wasserman 2004)
- Average log-likelihood ratio (ALLR) (Wang and Stormo 2003)
- Lower limit ALLR (ALLR_LL) (Mahony et al. 2007)
- Bhattacharyya coefficient (BHAT) (Bhattacharyya 1943)

Comparisons are calculated between two motifs at a time. All possible alignments are scored, and the best score is reported. In an alignment scores are calculated individually between columns. How those scores are combined to generate the final alignment scores depends on `score.strat`.

See the "Motif comparisons and P-values" vignette for a description of the various metrics. Note that PCC, WPCC, SW, ALLR, ALLR_LL and BHAT are similarities; higher values mean more similar motifs. For the remaining metrics, values closer to zero represent more similar motifs.

Small pseudocounts are automatically added when one of the following methods is used: KL, ALLR, ALLR_LL, IS. This is avoid zeros in the calculations.

Calculating P-values:

To note regarding p-values: P-values are pre-computed using the `make_DBscores()` function. If not given, then uses a set of internal precomputed P-values from the JASPAR2018 CORE motifs. These precalculated scores are dependent on the length of the motifs being compared. This takes into account that comparing small motifs with larger motifs leads to higher scores, since the probability of finding a higher scoring alignment is higher.

The default P-values have been precalculated for regular DNA motifs. They are of little use for motifs with a different number of alphabet letters (or even the `multifreq` slot).

Value

A list of matrices containing the similarity scores between motifs from different groups of sequences. The order of comparison is as follows, with first element representing the rows and second element representing the columns of the matrix:

- 1. **Common motifs comparison:** Common seqs from reference (1) <-> comparison (2)
- 2. **Unique motifs comparison:** Unique seqs from reference (1) <-> comparison (2)
- 3. **Cross motifs comparison 1:** Unique seqs from reference (1) <-> comparison (1)
- 4. **Cross motifs comparison 2:** Unique seqs from comparison (2) <-> reference (1)

The list is repeated for each set of comparison groups in input.

Examples

```
if (memes::meme_is_installed()) {
    data("CTCF_TIP_peaks", package = "MotifPeeker")
    data("CTCF_ChIP_peaks", package = "MotifPeeker")

    if (requireNamespace("BSgenome.Hsapiens.UCSC.hg38")) {
        genome_build <-
            BSgenome.Hsapiens.UCSC.hg38::BSgenome.Hsapiens.UCSC.hg38
        segregated_peaks <- segregate_seqs(CTCF_TIP_peaks, CTCF_ChIP_peaks)
        denovo_motifs <- denovo_motifs(unlist(segregated_peaks),
                            trim_seq_width = 50,
```

```
                            genome_build = genome_build,
                            discover_motifs_count = 1,
                            filter_n = 6,
                            maxw = 8,
                            minw = 8,
                            out_dir = tempdir())
        similarity_matrices <- motif_similarity(denovo_motifs)
        print(similarity_matrices)
    }
}
```

`normalise_paths` *Apply* `normalizePath` *to a list of paths*

Description

Apply `normalizePath` to a list of paths

Usage

```
normalise_paths(path_list)
```

Arguments

`path_list`	A list of paths.

Value

A list of normalised paths or the input as is if contents are not a character.

`plot_enrichment_individual`
Plot motif-enrichment for individual experiments

Description

Visualises the result from `get_df_enrichment` for a single motif by producing a `plotly` bar plot with the motif enrichment comparisons for one comparison dataset pair.

Usage

```
plot_enrichment_individual(
  result,
  enrichment_df,
  comparison_i,
  motif_i,
  label_colours,
  reference_index = 1,
  html_tags = TRUE
)
```

Arguments

result	A list with the following elements: **peaks** A list of peak files generated using read_peak_file. **alignments** A list of alignment files. **exp_type** A character vector of experiment types. **exp_labels** A character vector of experiment labels. **read_count** A numeric vector of read counts. **peak_count** A numeric vector of peak counts.
enrichment_df	A data frame containing the motif enrichment results, produced using get_df_enrichment.
comparison_i	The index of the comparison dataset to plot.
motif_i	The index of the motif to plot.
label_colours	A vector with colours (valid names or hex codes) to use for "No" and "Yes" bar segments.
reference_index	An integer specifying the index of the peak file to use as the reference dataset for comparison. Indexing starts from 1. (default = 1)
html_tags	Logical. If TRUE, returns the plot as a tagList object.

Value

A plotly object with the peak motif enrichment data. If html_tags is TRUE, the function returns a tagList object instead.

See Also

Other plot functions: plot_enrichment_overall(), plot_motif_comparison()

plot_enrichment_overall *Plot motif-enrichment for all experiments*

Description

Visualises the result from get_df_enrichment for a single motif by producing a plotly bar plot with the motif enrichment comparisons for all the comparison dataset pair.

Usage

```
plot_enrichment_overall(
  enrichment_df,
  motif_i,
  label_colours,
  reference_label,
  html_tags = TRUE
)
```

Arguments

enrichment_df	A data frame containing the motif enrichment results, produced using get_df_enrichment.
motif_i	The index of the motif to plot.
label_colours	A vector with colours (valid names or hex codes) to use for "No" and "Yes" bar segments.
reference_label	The label of the reference experiment.
html_tags	Logical. If TRUE, returns the plot as a tagList object.

Value

A list of plotly objects with the peak motif enrichment data. If html_tags is TRUE, the function returns a list of tagList objects instead. The two plots in the list are named as follows:

$count_plt y-axis represents the number of peaks.

$perc_plt y-axis represents the percentage of peaks.

See Also

Other plot functions: plot_enrichment_individual(), plot_motif_comparison()

`plot_motif_comparison` *Produce heat maps of motif similarity matrices*

Description

Produce heat maps of motif similarity matrices

Usage

```
plot_motif_comparison(
  comparison_matrices,
  exp_labels,
  height = NULL,
  width = NULL,
  html_tags = TRUE
)
```

Arguments

`comparison_matrices`	Output from `compare_motifs` for one comparison pair (Four matrices).
`exp_labels`	Labels for the reference and comparison experiments respectively.
`width, height`	The width and height of the output htmlwidget, or the output file if exporting to png/pdf/etc. Presumed to be in pixels, but if a plotly internal function decides it's in other units you may end up with a huge file! Default is 800x500 when exporting to a file, and 100 as a htmlwidget.
`html_tags`	Logical. If TRUE, returns the plot as a `tagList` object.

Value

A list of individual heat maps for the four matrices. If `html_tags` is `TRUE`, the output will be wrapped in HTML tags.

See Also

Other plot functions: `plot_enrichment_individual()`, `plot_enrichment_overall()`

`pretty_number` *Convert numbers to more readable strings*

Description

Format raw numbers to more readable strings. For example, 1000 will be converted to "1K". Supported suffixes are "K", "M", and "B".

Usage

```
pretty_number(x, decimal_digits = 2)
```

Arguments

`x`	A number.
`decimal_digits`	Number of decimal digits to round to.

Value

A character string of the formatted number. `NA` is returned as "NA".

Examples

```
print(MotifPeeker:::pretty_number(134999))
```

`print_denovo_sections` *Print denovo motif enrichment* `datatable` *and download buttons for related files.*

Description

Print denovo motif enrichment `datatable` and download buttons for related files.

Usage

```
print_denovo_sections(
  motif_list,
  similar_motifs,
  segregated_peaks,
  indices,
  jaspar_link = FALSE,
  download_buttons = NULL
)
```

Arguments

motif_list	A list of motifs discovered by find_motifs, for one comparison pair.
similar_motifs	A list of similar motifs discovered using motif_similarity, for one comparison pair.
segregated_peaks	A list of peaks segregated by common and unique groups, for one comparison pair.
indices	A list of indices to print the datatable and download buttons for.
jaspar_link	A logical indicating whether to include a link to the JASPAR database for the motifs. Only set to TRUE if the motifs are in JASPAR format (example: "MA1930.1").
download_buttons	Embed download buttons generated using get_download_buttons. If set to NULL, no download buttons will be added.

Value

Null

See Also

Other datatable functions: dt_enrichment_individual()

---

print_DT *Print DT table*

Description

Print DT table

Usage

```
print_DT(df, ..., html_tags = FALSE, extra = FALSE)
```

Arguments

df	Dataframe/tibble to be printed.
...	Arguments passed on to DT::datatable data a data object (either a matrix or a data frame) options a list of initialization options (see https://datatables.net/reference/option/); the character options wrapped in JS() will be treated as literal JavaScript code instead of normal character strings; you can also set options globally via options(DT.options = list(...)), and global options will be merged into this options argument if set class the CSS class(es) of the table; see https://datatables.net/manual/styling/classes

`callback` the body of a JavaScript callback function with the argument `table` to be applied to the DataTables instance (i.e. `table`)

`rownames` `TRUE` (show row names) or `FALSE` (hide row names) or a character vector of row names; by default, the row names are displayed in the first column of the table if exist (not `NULL`)

`colnames` if missing, the column names of the data; otherwise it can be an unnamed character vector of names you want to show in the table header instead of the default data column names; alternatively, you can provide a *named* numeric or character vector of the form `'newName1' = i1, 'newName2' = i2` or `c('newName1' = 'oldName1', 'newName2' = 'oldName2', ...)`, where `newName` is the new name you want to show in the table, and `i` or `oldName` is the index of the current column name

`container` a sketch of the HTML table to be filled with data cells; by default, it is generated from `htmltools::tags$table()` with a table header consisting of the column names of the data

`caption` the table caption; a character vector or a tag object generated from `htmltools::tags$caption()`

`filter` whether/where to use column filters; `none`: no filters; `bottom/top`: put column filters at the bottom/top of the table; range sliders are used to filter numeric/date/time columns, select lists are used for factor columns, and text input boxes are used for character columns; if you want more control over the styles of filters, you can provide a named list to this argument; see Details for more

`escape` whether to escape HTML entities in the table: `TRUE` means to escape the whole table, and `FALSE` means not to escape it; alternatively, you can specify numeric column indices or column names to indicate which columns to escape, e.g. `1:5` (the first 5 columns), `c(1, 3, 4)`, or `c(-1, -3)` (all columns except the first and third), or `c('Species', 'Sepal.Length')`; since the row names take the first column to display, you should add the numeric column indices by one when using `rownames`

`style` either `'auto'`, `'default'`, `'bootstrap'`, or `'bootstrap4'`. If `'auto'`, and a **bslib** theme is currently active, then bootstrap styling is used in a way that "just works" for the active theme. Otherwise, DataTables `'default'` styling is used. If set explicitly to `'bootstrap'` or `'bootstrap4'`, one must take care to ensure Bootstrap's HTML dependencies (as well as Bootswatch themes, if desired) are included on the page. Note, when set explicitly, it's the user's responsibility to ensure that only one unique 'style' value is used on the same page, if multiple DT tables exist, as different styling resources may conflict with each other.

`width,height` Width/Height in pixels (optional, defaults to automatic sizing)

`elementId` An id for the widget (a random string by default).

`fillContainer` `TRUE` to configure the table to automatically fill it's containing element. If the table can't fit fully into it's container then vertical and/or horizontal scrolling of the table cells will occur.

`autoHideNavigation` `TRUE` to automatically hide navigational UI (only display the table body) when the number of total records is less than the page

size. Note, it only works on the client-side processing mode and the 'pageLength' option should be provided explicitly.

`selection` the row/column selection mode (single or multiple selection or disable selection) when a table widget is rendered in a Shiny app; alternatively, you can use a list of the form `list(mode = 'multiple', selected = c(1, 3, 8), target = 'row', selectable = c(-2, -3))` to pre-select rows and control the selectable range; the element `target` in the list can be `'column'` to enable column selection, or `'row+column'` to make it possible to select both rows and columns (click on the footer to select columns), or `'cell'` to select cells. See details section for more info.

`extensions` a character vector of the names of the DataTables extensions (https://datatables.net/extensions/index)

`plugins` a character vector of the names of DataTables plug-ins (https://rstudio.github.io/DT/plugins.html). Note that only those plugins supported by the `DT` package can be used here. You can see the available plugins by calling `DT:::available_plugins()`

`editable` `FALSE` to disable the table editor, or `TRUE` (or `"cell"`) to enable editing a single cell. Alternatively, you can set it to `"row"` to be able to edit a row, or `"column"` to edit a column, or `"all"` to edit all cells on the current page of the table. In all modes, start editing by doubleclicking on a cell. This argument can also be a list of the form `list(target = TARGET, disable = list(columns = INDICES))`, where `TARGET` can be `"cell"`, `"row"`, `"column"`, or `"all"`, and `INDICES` is an integer vector of column indices. Use the list form if you want to disable editing certain columns. You can also restrict the editing to accept only numbers by setting this argument to a list of the form `list(target = TARGET, numeric = INDICES)` where `INDICES` can be the vector of the indices of the columns for which you want to restrict the editing to numbers or `"all"` to restrict the editing to numbers for all columns. If you don't set `numeric`, then the editing is restricted to numbers for all numeric columns; set `numeric = "none"` to disable this behavior. It is also possible to edit the cells in text areas, which are useful for large contents. For that, set the `editable` argument to a list of the form `list(target = TARGET, area = INDICES)` where `INDICES` can be the vector of the indices of the columns for which you want the text areas, or `"all"` if you want the text areas for all columns. Of course, you can request the numeric editing for some columns and the text areas for some other columns by setting `editable` to a list of the form `list(target = TARGET, numeric = INDICES1, area = INDICES2)`. Finally, you can edit date cells with a calendar with `list(target = TARGET, date = INDICES)`; the target columns must have the `Date` type. If you don't set `date` in the `editable` list, the editing with the calendar is automatically set for all `Date` columns.

`html_tags` Logical. If TRUE, returns the table as a `tagList` object.

`extra` Logical. If TRUE, adds extra options like search to the datatable.

Value

A DT object suitable to be used with `print()`.

print_labels *Print the labels of the reference and comparison experiments*

Description

Print the labels of the reference and comparison experiments

Usage

```
print_labels(
  exp_labels,
  reference_index,
  comparison_index,
  header_type,
  read_counts = NULL
)
```

Arguments

exp_labels	A character vector of experiment labels.
reference_index	The index of the reference experiment.
comparison_index	The index of the comparison experiment.
header_type	Label for the section to print the header for. Options are: "known_motif" and "denovo_motif".
read_counts	A numeric vector of read counts for each experiment. (optional)

Value

String with the labels of the reference and comparison experiments.

random_string *Generate a random string*

Description

Generate a random string

Usage

```
random_string(length)
```

Arguments

length	The length of the random string to generate.

Value

A random string of the specified length.

---

read_motif_file *Read a motif file*

Description

read_motif_file() reads a motif file and converts to a PWM. The function supports multiple motif formats, including "homer", "jaspar", "meme", "transfac" and "uniprobe".

Usage

```
read_motif_file(
  motif_file,
  motif_id = "Unknown",
  file_format = "auto",
  verbose = FALSE
)
```

Arguments

motif_file	Path to a motif file or a universalmotif-class object.
motif_id	ID of the motif (e.g. "MA1930.1").
file_format	Character string specifying the format of the motif file. The options are "homer", "jaspar", "meme", "transfac" and "uniprobe"
verbose	A logical indicating whether to print messages.

Value

A universalmotif motif object.

Examples

```
motif_file <- system.file("extdata",
                          "motif_MA1930.2.jaspar",
                          package = "MotifPeeker")
res <- read_motif_file(motif_file = motif_file,
                       motif_id = "MA1930.2",
                       file_format = "jaspar")
print(res)
```

read_peak_file *Read MACS2/3 narrowPeak or SEACR BED peak file*

Description

This function reads a MACS2/3 narrowPeak or SEACR BED peak file and returns a GRanges object with the peak coordinates and summit.

Usage

```
read_peak_file(peak_file, file_format = "auto", verbose = FALSE)
```

Arguments

peak_file	A character string with the path to the peak file, or a GRanges object created using read_peak_file().
file_format	A character string specifying the format of the peak file. • "narrowpeak": MACS2/3 narrowPeak format. • "bed": SEACR BED format.
verbose	A logical indicating whether to print messages.

Details

The *summit* column is the absolute genomic position of the peak, which is relative to the start position of the sequence range. For SEACR BED files, the *summit* column is calculated as the midpoint of the max signal region.

Value

A GRanges-class object with the peak coordinates and summit.

See Also

GRanges-class for more information on GRanges objects.

Examples

```
macs3_peak_file <- system.file("extdata", "CTCF_ChIP_peaks.narrowPeak",
package = "MotifPeeker")
macs3_peak_read <- read_peak_file(macs3_peak_file)
macs3_peak_read
```

`read_peak_file_macs` *Read MACS2/3 narrowPeak peak file*

Description

This function reads a MACS2/3 narrowPeak peak file and returns a GRanges object with the peak coordinates and summit.

Usage

```
read_peak_file_macs(peak_file)
```

Arguments

`peak_file`	A character string with the path to the peak file, or a GRanges object created using `read_peak_file()`.

Details

The *summit* column is the absolute genomic position of the peak, which is relative to the start position of the sequence range. For SEACR BED files, the *summit* column is calculated as the midpoint of the max signal region.

Value

A GRanges-class object with the peak coordinates and summit.

See Also

GRanges-class for more information on GRanges objects.

Examples

```
macs3_peak_file <- system.file("extdata", "CTCF_ChIP_peaks.narrowPeak",
package = "MotifPeeker")
macs3_peak_read <- read_peak_file(macs3_peak_file)
macs3_peak_read
```

read_peak_file_seacr *Read SEACR BED peak file*

Description

This function reads a SEACR BED peak file and returns a GRanges object with the peak coordinates and summit.

Usage

```
read_peak_file_seacr(peak_file)
```

Arguments

peak_file	A character string with the path to the peak file, or a GRanges object created using read_peak_file().

Details

The *summit* column is the absolute genomic position of the peak, which is relative to the start position of the sequence range. For SEACR BED files, the *summit* column is calculated as the midpoint of the max signal region.

Value

A GRanges-class object with the peak coordinates and summit.

See Also

GRanges-class for more information on GRanges objects.

report_command *Report command*

Description

Reconstruct the MotifPeeker command from the parameters used to generate the HTML report.

Usage

```
report_command(params)
```

Arguments

params	A list of parameters used to generate the HTML report.

Value

A character string containing the reconstructed MotifPeeker command.

Examples

```
MotifPeeker:::report_command(params = list(
   alignment_files = c("file1.bam", "file2.bam"),
   exp_labels = c("exp1", "exp2"),
   genome_build = "hg19"))
```

---

report_header *Report header*

Description

Credit: This function was adapted from the *EpiCompare* package.

Usage

```
report_header()
```

Details

Generate a header for MotifPeeker reports generated using the *MotifPeeker.Rmd* template.

Value

Header string to be rendering within Rmarkdown file.

Examples

```
MotifPeeker:::report_header()
```

---

save_peak_file *Minimally save a peak object to a file (BED4)*

Description

This function saves a peak object to a file in BED4 format. The included columns are: `chr`, `start`, `end`, and `name`. Since no strand data is being included, it is recommended to use this function only for peak objects that do not have strand information.

Usage

```
save_peak_file(
  peak_obj,
  save = TRUE,
  filename = random_string(10),
  out_dir = tempdir()
)
```

Arguments

peak_obj	A GRanges object with the peak coordinates. Must include columns: seqnames, start, end, and name.
save	A logical indicating whether to save the peak object to a file.
filename	A character string of the file name. If the file extension is not .bed, a warning is issued and the extension is appended. Alternatively, if the file name does not have an extension, .bed is appended. (default = random string)
out_dir	A character string of the output directory. (default = tempdir())

Value

If save = FALSE, a data frame with the peak coordinates. If save = TRUE, the path to the saved file.

Examples

```
data("CTCF_ChIP_peaks", package = "MotifPeeker")

out <- save_peak_file(CTCF_ChIP_peaks, save = TRUE, "test_peak_file.bed")
print(out)
```

---

segregate_seqs *Segregate input sequences into common and unique groups*

Description

This function takes two sets of sequences and segregates them into common and unique sequences. The common sequences are sequences that are present in both sets of sequences. The unique sequences are sequences that are present in only one of the sets of sequences.

Usage

```
segregate_seqs(seqs1, seqs2)
```

Arguments

seqs1	A set of sequences (GRanges object)
seqs2	A set of sequences (GRanges object)

Details

Sequences are considered common if their base pairs align in any position, even if they vary in length. Consequently, while the number of common sequences remains consistent between both sets, but the length and composition of these sequences may differ. As a result, the function returns distinct sets of common sequences for each input set of sequences.

Value

A list containing the common sequences and unique sequences for each set of sequences. The list contains the following `GRanges` objects:

- `common_seqs1`: Common sequences in `seqs1`
- `common_seqs2`: Common sequences in `seqs2`
- `unique_seqs1`: Unique sequences in `seqs1`
- `unique_seqs2`: Unique sequences in `seqs2`

See Also

findOverlaps

Examples

```
data("CTCF_ChIP_peaks", package = "MotifPeeker")
data("CTCF_TIP_peaks", package = "MotifPeeker")

seqs1 <- CTCF_ChIP_peaks
seqs2 <- CTCF_TIP_peaks
res <- segregate_seqs(seqs1, seqs2)
print(res)
```

---

summit_to_motif *Calculate the distance between peak summits and motifs*

Description

`summit_to_motif()` calculates the distance between each motif and its nearest peak summit. `runFimo` from the `memes` package is used to recover the locations of each motif.

Usage

```
summit_to_motif(
  peak_input,
  motif,
  fp_rate = 0.05,
  genome_build,
  out_dir = tempdir(),
```

```
  meme_path = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

peak_input	Either a path to the narrowPeak file or a GRanges peak object generated by `read_peak_file()`.
motif	An object of class `universalmotif`.
fp_rate	The desired false-positive rate. A p-value threshold will be selected based on this value. The default false-positive rate is 0.05.
genome_build	The genome build that the peak sequences should be derived from.
out_dir	Location to save the 0-order background file. By default, the background file will be written to a temporary directory.
meme_path	path to "meme/bin/" (default: `NULL`). Will use default search behavior as described in `check_meme_install()` if unset.
verbose	A logical indicating whether to print verbose messages while running the function. (default = FALSE)
...	Arguments passed on to `memes::runFimo`

`parse_genomic_coord logical(1)` whether to parse genomic position from fasta headers. Fasta headers must be UCSC format positions (ie "chr:start-end"), but base 1 indexed (GRanges format). If names of fasta entries are genomic coordinates and parse_genomic_coord == TRUE, results will contain genomic coordinates of motif matches, otherwise FIMO will return relative coordinates (i.e. positions from 1 to length of the fasta entry).

`skip_matched_sequence logical(1)` whether or not to include the DNA sequence of the match. Default: `FALSE`. Note: jobs will complete faster if set to `TRUE`. `add_sequence()` can be used to lookup the sequence after data import if `parse_genomic_coord` is `TRUE`, so setting this flag is not strictly needed.

`max_strand` if match is found on both strands, only report strand with best match (default: TRUE).

`text logical(1)` (default: `TRUE`). No output files will be created on the filesystem. The results are unsorted and no q-values are computed. This setting allows fast searches on very large inputs. When set to `FALSE` FIMO will discard 50% of the lower significance matches if >100,000 matches are detected. `text = FALSE` will also incur a performance penalty because it must first read a file to disk, then read it into memory. For these reasons, I suggest keeping `text = TRUE`.

`silent logical(1)` whether to suppress stdout/stderr printing to console (default: TRUE). If the command is failing or giving unexpected output, setting `silent = FALSE` can aid troubleshooting.

Details

To calculate the p-value threshold for a desired false-positive rate, we use the approximate formula:

$$p \approx \frac{fp\_rate}{2 \times \text{average peak width}}$$

(Dervied from FIMO documentation)

Value

A list containing an expanded GRanges peak object with metadata columns relating to motif positions along with a vector of summit-to-motif distances for each valid peak.

See Also

runAme

Examples

```
if (memes::meme_is_installed()) {
data("CTCF_TIP_peaks", package = "MotifPeeker")
data("motif_MA1102.3", package = "MotifPeeker")

res <- summit_to_motif(
    peak_input = CTCF_TIP_peaks,
    motif = motif_MA1102.3,
    fp_rate = 5e-02,
    genome_build = BSgenome.Hsapiens.UCSC.hg38::BSgenome.Hsapiens.UCSC.hg38
)
print(res)
}
```

---

to_plotly *Convert ggplot2 objects to plotly*

Description

Convert ggplot2 objects to plotly

Usage

```
to_plotly(p, html_tags = TRUE, tooltip = "text", ...)
```

Arguments

p	ggplot2 object
html_tags	Logical. If TRUE, returns the plot as a tagList object.
tooltip	Character. The tooltip to display. Default is "text".
...	Arguments passed on to plotly::ggplotly width Width of the plot in pixels (optional, defaults to automatic sizing). height Height of the plot in pixels (optional, defaults to automatic sizing). dynamicTicks should plotly.js dynamically generate axis tick labels? Dynamic ticks are useful for updating ticks in response to zoom/pan interactions; however, they can not always reproduce labels as they would appear in the static ggplot2 image. layerData data from which layer should be returned? originalData should the "original" or "scaled" data be returned? source a character string of length 1. Match the value of this string with the source argument in event_data() to retrieve the event data corresponding to a specific plot (shiny apps can have multiple plots).

Value

A plotly object.

See Also

ggplotly

Examples

```
x <- data.frame(a = c(1,2,3), b = c(2,3,4))
p <- ggplot2::ggplot(x, ggplot2::aes(x = a, y = b)) + ggplot2::geom_point()
MotifPeeker:::to_plotly(p, html_tags = FALSE)
```

---

trim_seqs *Trim sequences to a specified width around the summit*

Description

Trim sequences to a specified width around the summit

Usage

```
trim_seqs(peaks, peak_width, genome_build, respect_bounds = TRUE)
```

Arguments

peaks	A GRanges object created using read_peak_file().
peak_width	Total expected width of the peak.
genome_build	The genome build that the peak sequences should be derived from.
respect_bounds	Logical indicating whether the peak width should be respected when trimming sequences. (default = TRUE) If TRUE, the trimmed sequences will not extend beyond the peak boundaries.

Value

A GRanges object with the trimmed sequences. The sequences are guaranteed to not exceed the peak width + 1 (peak width + the summit base).

Examples

```
data("CTCF_TIP_peaks", package = "MotifPeeker")
peaks <- CTCF_TIP_peaks
genome_build <- BSgenome.Hsapiens.UCSC.hg38::BSgenome.Hsapiens.UCSC.hg38

trimmed_seqs <- MotifPeeker:::trim_seqs(peaks, peak_width = 100,
                        genome_build = genome_build)
summary(GenomicRanges::width(trimmed_seqs))
```

---

use_cache *Check, add and access files in cache*

Description

Query local BiocFileCache to get cached version of a file and add them if they do not exist.

Usage

```
use_cache(url, verbose = FALSE)
```

Arguments

url	A character string specifying the URL of the file to check for.
verbose	A logical indicating whether to print verbose messages while running the function. (default = FALSE)

Value

A character string specifying the path to the cached file.

`%>%` *Pipe operator*

Description

See `magrittr::%>%` for details.

Usage

```
lhs %>% rhs
```

Arguments

`lhs`	A value or the magrittr placeholder.
`rhs`	A function call using the magrittr semantics.

Details

Generated by `use_pipe`. Modified to import from `dplyr` instead of `magrittr`.

Value

The result of calling ‘rhs(lhs)‘.

Examples

```
seq_len(10) %>% sum
```

Index

∗ **datasets**
CTCF_ChIP_peaks, 9
CTCF_TIP_peaks, 10
motif_MA1102.3, 30
motif_MA1930.2, 30
∗ **datatable functions**
dt_enrichment_individual, 13
print_denovo_sections, 38
∗ **generate data.frames**
get_df_distances, 17
get_df_enrichment, 19
∗ **internal**
%>%, 54
bpapply, 3
check_dep, 5
check_duplicates, 6
check_input, 8
confirm_meme_install, 9
download_button, 12
dt_enrichment_individual, 13
filter_repeats, 14
format_exptype, 16
get_download_buttons, 21
link_JASPAR, 23
markov_background_model, 23
messager, 24
normalise_paths, 34
plot_enrichment_individual, 34
plot_enrichment_overall, 36
plot_motif_comparison, 37
pretty_number, 38
print_denovo_sections, 38
print_DT, 39
print_labels, 42
random_string, 42
read_peak_file_macs, 45
read_peak_file_seacr, 46
report_command, 46
report_header, 47
to_plotly, 51
trim_seqs, 52
use_cache, 53
∗ **plot functions**
plot_enrichment_individual, 34
plot_enrichment_overall, 36
plot_motif_comparison, 37
%>%, *54*, 54

average_ic(), *32*

BamFile, *25*
BiocParallel, *3, 4*
BiocParallel::bplapply, *4*
BiocParallel::bpmapply, *4*
bpapply, 3
bpoptions, *4*
BSgenome-class, *7*

calc_frip, 4
check_dep, 5
check_duplicates, 6
check_ENCODE, 6
check_genome_build, 7
check_input, 8
check_JASPAR, 8
check_meme_install, *9*
compare_motifs, *31*, *37*
compare_motifs(), *32*
confirm_meme_install, 9
convert_motifs(), *31*
convert_type(), *32*
CTCF_ChIP_peaks, 9
CTCF_TIP_peaks, 10

datatable, *13*, *38*
denovo_motifs, 10, *15*, *31*
download_button, 12
download_file, *13*
downloadthis::download_file, *13*

DT::datatable, *39*
dt_enrichment_individual, 13, *39*

event_data(), *52*

filter_repeats, 14
find_motifs, 15, *39*
findOverlaps, *49*
format_exptype, 16

get_df_distances, 17, *20*
get_df_enrichment, *14*, *18*, 19, *34–36*
get_download_buttons, 21, *39*
get_JASPARCORE, 22
ggplotly, *52*
GRanges, *11*
GRanges-class, *44–46*

JS, *39*

link_JASPAR, 23

make_DBscores(), *33*
markov_background_model, 23
memes, *28*
memes::runAme, *29*
memes::runFimo, *50*
message, *24*
messager, 24
motif_enrichment, 28
motif_MA1102.3, 30
motif_MA1930.2, 30
motif_similarity, 31, *39*
MotifPeeker, 24, *30*, *46, 47*

normalise_paths, 34
normalizePath, *34*

options, *39*

plot_enrichment_individual, 34, *36, 37*
plot_enrichment_overall, *35*, 36, *37*
plot_motif_comparison, *35, 36*, 37
plotly::ggplotly, *52*
pretty_number, 38
print_denovo_sections, *14*, 38
print_DT, 39
print_labels, 42

random_string, 42
read_motif_file, 43
read_peak_file, *13*, *17*, *19*, *25*, *35*, 44
read_peak_file(), *44–46*, *53*
read_peak_file_macs, 45
read_peak_file_seacr, 46
report_command, 46
report_header, 47
runAme, *29*, *51*
runStreme, *14*

save_peak_file, 47
segregate_seqs, *19*, *22*, 48
simplify2array, *4*
summit_to_motif, 49

tag, *13*
to_plotly, 51
trim_seqs, 52

universalmotif, *12*
universalmotif::compare_motifs, *31*
use_cache, 53
use_pipe, *54*