Package 'PMScanR'

February 28, 2026

Title Protein motifs analysis and visualisation

Version 1.0.1

Description Provides tools for large-scale protein motif analysis and visualization in R. PMScanR facilitates the identification of motifs using external tools like PROSITE's ps_scan (handling necessary file downloads and execution) and enables downstream analysis of results. Key features include parsing scan outputs, converting formats (e.g., to GFF-like structures), generating motif occurrence matrices, and creating informative visualizations such as heatmaps, sequence logos (via seqLogo/ggseqlogo). The package also offers an optional Shiny-based graphical user interface for interactive analysis, aiming to streamline the process of exploring motif patterns across multiple protein sequences.

URL https://github.com/prodakt/PMScanR

BugReports https://github.com/prodakt/PMScanR/issues

License GPL-3

Encoding UTF-8

LazyData false

RoxygenNote 7.3.3

VignetteBuilder knitr

biocViews MotifDiscovery, Visualization

Imports dplyr (>= 1.1.0), shiny, bslib, shinyFiles, plotly, rtracklayer, reshape2, ggseqlogo, ggplot2, seqinr, magrittr, rlang, utils, stringr, BiocFileCache

Suggests BiocStyle, knitr, seqLogo, rmarkdown, testthat (>= 3.0.0)

SystemRequirements Perl

Config/testthat/edition 3

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

git_branch RELEASE_3_22

git_last_commit c6ffc74

git_last_commit_date 2025-11-28

Repository Bioconductor 3.22

Date/Publication 2026-02-27

Author Jan Pawel Jastrzebski [aut, cre] (ORCID:
<https://orcid.org/0000-0001-8699-7742>),
Monika Gawronska [ctb] (ORCID: <https://orcid.org/0009-0001-2677-6371>),
Wiktor Babis [ctb] (ORCID: <https://orcid.org/0009-0006-3648-3413>),
Miriana Quaranta [ctb] (ORCID: <https://orcid.org/0009-0003-0855-485X>),
Damian Czopek [ctb, aut] (ORCID:
<https://orcid.org/0009-0005-3471-4866>)

Maintainer Jan Pawel Jastrzebski <bioinformatyka@gmail.com>

Contents

`PMScanR-package` *PMScanR: Protein motifs analysis and visualisation*

Description

Provides tools for large-scale protein motif analysis and visualization in R. PMScanR facilitates the identification of motifs using external tools like PROSITE's ps_scan (handling necessary file downloads and execution) and enables downstream analysis of results. Key features include parsing scan outputs, converting formats (e.g., to GFF-like structures), generating motif occurrence matrices, and creating informative visualizations such as heatmaps, sequence logos (via seqLogo/ggseqlogo). The package also offers an optional Shiny-based graphical user interface for interactive analysis, aiming to streamline the process of exploring motif patterns across multiple protein sequences.

Author(s)

Maintainer: Jan Pawel Jastrzebski <bioinformatyka@gmail.com> (ORCID)

Authors:

- Damian Czopek <dcwmpl@gmail.com> (ORCID) [contributor]

Other contributors:

- Monika Gawronska <gawronska572@gmail.com> (ORCID) [contributor]
- Wiktor Babis <wiktorbabis@gmail.com> (ORCID) [contributor]
- Miriana Quaranta <miriana.quaranta@uniroma1.it> (ORCID) [contributor]

See Also

Useful links:

- https://github.com/prodakt/PMScanR
- Report bugs at https://github.com/prodakt/PMScanR/issues

`extractProteinMotifs` *Extract protein motifs from GFF, PSA, or PROSITE text files*

Description

This function extracts protein motif sequences from various file formats output by PROSITE analysis tools. It automatically detects the format (GFF, PSA, or standard PROSITE scan output) and returns a list of sequences grouped by motif identifier.

Usage

```
extractProteinMotifs(file_path, format = "auto")
```

Arguments

`file_path`	A character string specifying the path to the input file.
`format`	A character string specifying the format: "auto" (default), "gff", "psa", or "scan" (for PROSITE text output).

Value

A list where keys are motif identifiers (e.g., "PS00001") and values are character vectors of the corresponding motif sequences found. Returns an empty list if no motifs/sequences are found.

Examples

```
# Example with PSA file
psa_file <- system.file("extdata", "out_Hb_psa.txt", package = "PMScanR")
if (nzchar(psa_file)) {
  motifs <- extractProteinMotifs(psa_file)
  # head(motifs$PS00005)
}
```

extractSegments *Extract sequence fragments from a list of sequences*

Description

This function iterates over a list of sequences and extracts a sub-sequence from each based on a specified start and end position.

Usage

```
extractSegments(sequences, from, to)
```

Arguments

sequences	A list of sequences, where each element is a vector of single characters. This is typically the output of ‘seqinr::read.fasta‘.
from	An integer specifying the starting position for the extraction.
to	An integer specifying the ending position for the extraction.

Value

A list representing the extracted sub-sequences. Sequences that were too short to have a fragment extracted are omitted from the list.

Examples

```
# Get the path to the example FASTA file
fasta_file <- system.file("extdata", "hemoglobins.fasta", package = "PMScanR")

if (nzchar(fasta_file)) {
  sequences <- seqinr::read.fasta(fasta_file, seqtype = "AA")
  segments <- extractSegments(sequences, from = 10, to = 20)
}
```

freqPie *Create a pie chart showing protein motif distribution*

Description

This function calculates the occurrences and percentages for each protein motif in the 'Name' column of a GFF-like data frame. It then creates a pie chart using ‘ggplot2‘ to visualize the distribution.

Usage

```
freqPie(data)
```

Arguments

data	A data frame in GFF format containing a column named 'Name' with the names of each protein motif.

Value

A ggplot object representing the pie chart.

Examples

```
# Create sample data frame similar to parsed GFF output
sample_data <- data.frame(
  seqid = rep(c("Seq1", "Seq2"), each = 5),
  source = rep("PROSITE", 10),
  type = rep("MOTIF", 10),
  start = sample(1:100, 10),
  end = sample(101:200, 10),
  score = runif(10),
  strand = sample(c("+", "-"), 10, replace = TRUE),
  phase = sample(0:2, 10, replace = TRUE),
  Name = sample(c("Zinc_finger", "EGF_domain", "Kinase_domain"), 10, replace = TRUE)
)

# Generate the pie chart
motif_pie_chart <- freqPie(sample_data)
# print(motif_pie_chart)
```

`gff2matrix` *Convert GFF to a binary occurrence matrix*

Description

This function takes a GFF data frame and converts it into a binary matrix, indicating the presence (1) or absence (0) of a feature in a sequence.

Usage

```
gff2matrix(input)
```

Arguments

`input`	A data frame containing GFF data, typically generated by 'rtracklayer::import.gff' and converted to a data frame. It must have 'type', 'start', 'end', and 'seqnames' columns.

Value

A matrix, where values are binary: '1' indicates the presence of a feature, and '0' indicates its absence.

Examples

```
gff_file <- system.file("extdata/out_Hb_gff.txt", package = "PMScanR")
if (nzchar(gff_file)) {
  gff_data <- as.data.frame(rtracklayer::import.gff(gff_file))
  motif_matrix <- gff2matrix(gff_data)
  # print(head(motif_matrix))
}
```

matrix2OP *Generate a occurrence plot from a matrix*

Description

This function generates a occurrence plot using the 'plotly' package. The occurrence plot highlights specific rows and columns provided by the user, while the rest of the matrix is dimmed. The function also adds grid lines to the occurrence plot for better readability.

Usage

```
matrix2OP(input, x = NULL, y = NULL)
```

Arguments

input	A matrix containing the data to be visualized in the occurrence plot
x	A character vector specifying the columns to highlight in the occurrence plot
y	A character vector specifying the rows to highlight in the occurrence plot

Value

A occurrence plot with highlighted specified rows and columns

Examples

```
# Create a sample matrix with row and column names
mat <- matrix(c(1, 0, 1, 0), 2, 2)
colnames(mat) <- c("Col1", "Col2")
rownames(mat) <- c("Row1", "Row2")
occurrence_plot <- matrix2OP(input = mat, x = "Col1", y = "Row1")
occurrence_plot
```

matrix2SquareOP *Generate a square occurrence plot from a matrix*

Description

This function generates a occurrence plot using 'plotly', ensuring the plot has a square aspect ratio. It highlights user-specified rows and columns.

Usage

```
matrix2SquareOP(input, x = NULL, y = NULL)
```

Arguments

input	A matrix containing the data to be visualized.
x	A character vector specifying the columns to highlight.
y	A character vector specifying the rows to highlight.

Value

A plotly heatmap object with a square layout.

Examples

```
# Create a sample matrix
mat <- matrix(c(1, 0, 1, 0), 2, 2)
colnames(mat) <- c("Col1", "Col2")
rownames(mat) <- c("Row1", "Row2")
square_occurrence_plot <- matrix2SquareOP(input = mat, x = "Col1", y = "Row1")
# To display in an interactive session:
# sq_heatmap
```

`readProsite` *Convert PROSITE format to a GFF-like Data Frame*

Description

This function parses a file from a PROSITE scan (standard output format) into a data frame. It handles multi-line sequence outputs and extracts information into a GFF-like structure compatible with rtracklayer imports.

Usage

```
readProsite(prosite_input)
```

Arguments

`prosite_input`	Path to the PROSITE scan output file.

Value

A data frame with columns approximating GFF fields plus additional PROSITE-specific information.

Examples

```
# Get path to example file
prosite_file <- system.file("extdata", "out_Hb_PROSITE.txt", package = "PMScanR")

if (nzchar(prosite_file) && file.exists(prosite_file)) {
  prosite_data <- readProsite(prosite_file)
  head(prosite_data)
}
```

readPsa *Parse a PSA (PROSITE Scan ASCII) File*

Description

This function reads a file in PSA format and converts it into a standardized, GFF-like data frame for downstream analysis. It ensures compatibility with data frames generated by rtracklayer::import.gff by matching column types and order.

Usage

```
readPsa(psa_file)
```

Arguments

psa_file	A character string specifying the path to the input PSA file.

Value

A data frame with a GFF-like structure.

Examples

```
# Get path to example file
psa_file <- system.file("extdata", "out_Hb_psa.txt", package = "PMScanR")

if (nzchar(psa_file) && file.exists(psa_file)) {
  psa_data <- readPsa(psa_file)
  head(psa_data)
}
```

runPMScanRShiny *Launch the PMScanR Shiny Application*

Description

Calling this function will launch the interactive graphical user interface for the PMScanR package.

Usage

```
runPMScanRShiny()
```

Details

This function sets a higher file upload size limit for Shiny and then launches the application, which is built using an internal UI function (‘buildUi‘) and server function (‘buildServer‘).

Value

This function is called for its side effect of launching the Shiny application and does not return a value.

Examples

```
if (interactive()) {
  # To run the app, simply call the function
  runPMScanRShiny()
}
```

---

runPsScan *Run PS-Scan with Flexible Configuration*

Description

This function runs the PROSITE ps_scan tool. It allows users to provide their own paths to required files (database, script, executable). If paths are not provided, it handles the downloading and caching of required executables and databases using BiocFileCache.

Usage

```
runPsScan(
  in_file,
  out_file,
  out_format = "scan",
  database_path = NULL,
  ps_scan_path = NULL,
  pfscan_path = NULL,
  os = NULL
)
```

Arguments

in_file	Path to the input file containing protein sequences.
out_file	Path for the output file where results will be saved.
out_format	The output format for ps_scan (e.g., 'gff', 'psa', 'scan').
database_path	(Optional) Path to the PROSITE database file ('prosite.dat'). If NULL, the file is retrieved from the internal cache.
ps_scan_path	(Optional) Path to the 'ps_scan.pl' script. If NULL, the file is retrieved from the internal cache.
pfscan_path	(Optional) Path to the 'pfscan' executable (or 'pfscan.exe'). If NULL, the file is retrieved from the internal cache (except on Mac, where it is optional).
os	The operating system ('WIN', 'LINUX', 'MAC'). If NULL, it is detected automatically.

Value

Invisibly returns the exit status of the ps_scan command. The primary output is the result file created at 'out_file'.

Index